PAT-NO: JP410275082A

DOCUMENT-IDENTIFIER: JP 10275082 A

TITLE: SYSTEM AND METHOD FOR REMOTE OBJECT ACCESS

PUBN-DATE: October 13, 1998

INVENTOR-INFORMATION:
NAME
YAMADA, TAKASHI

X/ation Request submitted 17 Feb 2004

ASSIGNEE-INFORMATION:
NAME COUNTRY
N T T DATA TSUSHIN KK N/A

APPL-NO: JP10020613

APPL-DATE: February 2, 1998

INT-CL (IPC): G06F009/44, G06F009/46 , G06F013/00

ABSTRACT:

PROBLEM TO BE SOLVED: To dynamically generate an object on a computer across a network by making use of a remote procedure(RPC) and to access it.

SOLUTION: A client computer 11 has a main program 3 and a conversion logic client stub 8. A server computer 12 has a conversion logic server library 9. In response to a generation request from the main program 3, the server library 9 generates an object 2, assign a unique object ID thereto, and stores this object ID and a pointer into a correspondence table 10 and also returns the object ID to the main program 3. Then, the main program 3 specifies the object ID and issues a process request and a deletion request to the object 2. The server library 9 acquires the pointer corresponding to the object ID from the correspondence table 10 and processes the object 2 that the pointer 2 indicates.

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－275082

(43)公開日　平成10年(1998)10月13日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/44	５３０	Ｇ０６Ｆ	9/44	５３０Ｍ
	9/46	３６０		9/46	３６０Ｂ
	13/00	３５７		13/00	３５７Ｚ

審査請求　未請求　請求項の数10　ＯＬ　（全 14 頁）

(21)出願番号　特願平10－20613

(22)出願日　平成10年(1998)２月２日

(31)優先権主張番号　特願平9－20747

(32)優先日　平９(1997)２月３日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000102728
エヌ・ティ・ティ・データ通信株式会社
東京都江東区豊洲三丁目３番３号

(72)発明者　山田　隆司
東京都江東区豊洲三丁目３番３号　エヌ・ティ・ティ・データ通信株式会社内

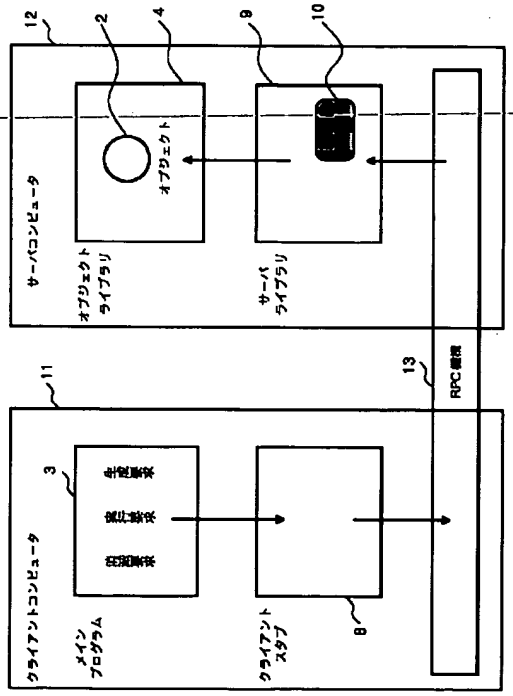
(74)代理人　弁理士　上村　輝之

(54)【発明の名称】　リモートオブジェクトアクセスシステム及び方法

(57)【要約】　（修正有）

【課題】　リモートプロシージャコール（ＲＰＣ）を利用してネットワークを隔てたコンピュータ上にオブジェクトを動的に生成し且つそれにアクセスする。

【解決手段】　クライアントコンピュータ１１はメインプログラム３と変換ロジッククライアントスタブ８とを有する。サーバコンピュータ１２は変換ロジックサーバライブラリ９を有する。メインプログラム３からの生成要求に応答して、サーバライブラリ９がオブジェクト２を生成し、これにユニークなオブジェクトＩＤを割り当て、このオブジェクトＩＤとポインタとを対応テーブル１０に格納し、かつ、そのオブジェクトＩＤをメインプログラム３に返却する。以後、メインプログラム３は、そのオブジェクトＩＤを指定してオブジェクト２への処理要求や消滅要求を発行する。サーバライブラリ９は、対応テーブル１０からオブジェクトＩＤに対応するポインタを取得し、そのポインタが指すオブジェクト２に対し処理を行う。



1

【特許請求の範囲】

【請求項1】　リモートプロシージャコール（ＲＰＣ）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムにおいて、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に応答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するＲＰＣクライアントスタブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記ＲＰＣクライアントスタブからの個々の関数の呼び出しに応答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するＲＰＣサーバライブラリを備え、

ＲＰＣクライアントスタブと前記ＲＰＣサーバライブラリとは、各サーバオブジェクトに割当てられたユニークなインスタンスＩＤを前記リモートプロシージャコールを通じてやりとりし、

前記ＲＰＣサーバライブラリは、各サーバオブジェクトのインスタンスＩＤをポインタに変換する手段を有することを特徴とするリモートオブジェクトアクセスシステム。

【請求項2】　リモートプロシージャコール（ＲＰＣ）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするための方法において、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に応答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するＲＰＣクライアントスタブとして、前記クライアントコンピュータが機能する過程と、

前記オブジェクトの生成及びアクセスのための関数を有して、前記ＲＰＣクライアントスタブからの個々の関数の呼び出しに応答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するＲＰＣサーバライブラリとして、前記サーバコンピュータが機能する過程と、

前記ＲＰＣクライアントスタブと前記ＲＰＣサーバライブラリとが、各サーバオブジェクトに割当てられたユニークなインスタンスＩＤを前記リモートプロシージャコールを通じてやりとりする過程と、

前記ＲＰＣサーバライブラリが、各サーバオブジェクトのインスタンスＩＤをポインタに変換する過程と、を有することを特徴とする方法。

2

【請求項3】　リモートプロシージャコール（ＲＰＣ）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムにおいて、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に応答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するＲＰＣクライアントスタブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記ＲＰＣクライアントスタブからの個々の関数の呼び出しに応答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するＲＰＣサーバライブラリを備え、

前記ＲＰＣサーバライブラリは、生成したサーバオブジェクトに対しユニークなインスタンスＩＤを割り当て、各サーバオブジェクトのインスタンスＩＤとポインタとの対応テーブルを保有し、且つ、前記生成したサーバオブジェクトに割り当てたインスタンスＩＤを前記ＲＰＣクライアントスタブへ返却し、

前記ＲＰＣクライアントスタブは、各サーバオブジェクトのアクセスのための関数呼び出しを発行する時、返却された各サーバオブジェクトのインスタンスＩＤを前記関数呼び出しに含ませる、ことを特徴とするリモートオブジェクトアクセスシステム。

【請求項4】　請求項3記載のシステムにおいて、

前記クライアントコンピュータが、前記オブジェクト生成のための要求に応答して生成されるクライアントオブジェクトを更に備え、

前記クライアントオブジェクトは、

(1)その生成時に、前記オブジェクト生成のための関数呼び出しを発行するよう前記ＲＰＣクライアントスタブに指示し、且つ、この生成関数の実行の結果返却されたインスタンスＩＤを前記ＲＰＣクライアントスタブより受けて保有し、さらに、

(2)前記オブジェクトのアクセスのための要求に応答して、前記保有したてインスタンスＩＤを使用して前記アクセスのための関数を発行するよう前記ＲＰＣクライアントスタブに指示することを特徴とするリモートオブジェクトアクセスシステム。

【請求項5】　請求項4記載のシステムにおいて、前記クライアントオブジェクトは、オブジェクト種類を示すオブジェクト種別ＩＤを更に有し、このクライアントオ

ブジェクトの生成時に、前記オブジェクト種別ＩＤを使用して前記オブジェクト生成のための関数呼び出しを発行するよう前記ＲＰＣクライアントスタブに指示し、
前記ＲＰＣクライアントスタブは、前記オブジェクト生成のための関数呼び出しを発行する時、前記オブジェクト種別ＩＤを前記関数呼び出しに含ませ、
前記ＲＰＣサーバライブラリは、オブジェクト種類とオブジェクト種別ＩＤとの対応テーブルを更に有し、前記オブジェクト生成のための関数呼び出しに応答して、前記オブジェクト種別ＩＤに対応したオブジェクト種類に属するオリジナルオブジェクトがサポートするメソッドを有したサーバオブジェクトを生成することを特徴とするリモートオブジェクトアクセスシステム。

【請求項６】　請求項５記載のシステムにおいて、
前記クライアントオブジェクトは、前記オリジナルオブジェクトから派生させて構築したものであって、前記オリジナルオブジェクトがサポートするメソッドに対する要求を受け付けることができ、
前記サーバオブジェクトは、前記オリジナルオブジェクトと所定の基本サーバオブジェクトとから派生させて構築したものであって、前記基本サーバオブジェクトがサポートするメソッドに対する要求を受け付けることができ、
前記クライアントオブジェクトは更に、前記オリジナルオブジェクトがサポートするメソッドに対する要求に応答して、そのメソッドに予め割当てられた処理種別を指定して、前記基本オブジェクトがサポートするメソッドに対する処理要求関数の呼出しを発行するよう前記ＲＰＣクライアントスタブに指示し、
更に、前記ＲＰＣサーバライブラリは、前記基本オブジェクトがサポートするメソッドに対する処理要求関数の呼出しに応答して、前記処理種別を指定して前記基本サーバオブジェクトがサポートするメソッドに対する要求を、前記サーバオブジェクトに発行し、
前記サーバオブジェクトは更に、前記基本サーバオブジェクトがサポートするメソッドに対する要求に応答して、オリジナルオブジェクトがサポートするメソッドの内の前記処理種別に対応するメソッドを実行することを特徴とするリモートオブジェクトアクセスシステム。

【請求項７】　リモートプロシージャコール（ＲＰＣ）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムであって、
前記クライアントコンピュータは、
前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に応答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するＲＰＣクライアントスタブを備え、

前記サーバコンピュータは、
前記オブジェクトの生成及びアクセスのための関数を有して、前記ＲＰＣクライアントスタブからの個々の関数の呼び出しに応答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するＲＰＣサーバライブラリを備え、
ＲＰＣクライアントスタブと前記ＲＰＣサーバライブラリとは、各サーバオブジェクトに割当てられたユニークなインスタンスＩＤを前記リモートプロシージャコールを通じてやりとりし、
前記ＲＰＣサーバライブラリは、各サーバオブジェクトのインスタンスＩＤをポインタに変換する手段を有するリモートオブジェクトアクセスシステムにおける、前記クライアントコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。

【請求項８】　リモートプロシージャコール（ＲＰＣ）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムであって、
前記クライアントコンピュータは、
前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に応答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するＲＰＣクライアントスタブを備え、
前記サーバコンピュータは、
前記オブジェクトの生成及びアクセスのための関数を有して、前記ＲＰＣクライアントスタブからの個々の関数の呼び出しに応答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するＲＰＣサーバライブラリを備え、
ＲＰＣクライアントスタブと前記ＲＰＣサーバライブラリとは、各サーバオブジェクトに割当てられたユニークなインスタンスＩＤを前記リモートプロシージャコールを通じてやりとりし、
前記ＲＰＣサーバライブラリは、各サーバオブジェクトのインスタンスＩＤをポインタに変換する手段を有するリモートオブジェクトアクセスシステムにおける、前記サーバコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。

【請求項９】　リモートプロシージャコール（ＲＰＣ）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトに

5

アクセスするためのリモートオブジェクトアクセスシステムであって、
前記クライアントコンピュータは、
前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に応答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するＲＰＣクライアントスタブを備え、
前記サーバコンピュータは、
前記オブジェクトの生成及びアクセスのための関数を有して、前記ＲＰＣクライアントスタブからの個々の関数の呼び出しに応答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するＲＰＣサーバライブラリを備え、
前記ＲＰＣサーバライブラリは、生成したサーバオブジェクトに対しユニークなインスタンスＩＤを割り当て、各サーバオブジェクトのインスタンスＩＤとポインタとの対応テーブルを保有し、且つ、前記生成したサーバオブジェクトに割り当てたインスタンスＩＤを前記ＲＰＣクライアントスタブへ返却し、
前記ＲＰＣクライアントスタブは、各サーバオブジェクトのアクセスのための関数呼び出しを発行する時、返却された各サーバオブジェクトのインスタンスＩＤを前記関数呼び出しに含ませる、リモートオブジェクトアクセスシステムにおける、前記クライアントコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。
【請求項１０】　リモートプロシージャコール（ＲＰＣ）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムであって、
前記クライアントコンピュータは、
前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に応答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するＲＰＣクライアントスタブを備え、
前記サーバコンピュータは、
前記オブジェクトの生成及びアクセスのための関数を有して、前記ＲＰＣクライアントスタブからの個々の関数の呼び出しに応答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するＲＰＣサーバライブラリを備え、
前記ＲＰＣサーバライブラリは、生成したサーバオブジェクトに対しユニークなインスタンスＩＤを割り当て、

6

各サーバオブジェクトのインスタンスＩＤとポインタとの対応テーブルを保有し、且つ、前記生成したサーバオブジェクトに割り当てたインスタンスＩＤを前記ＲＰＣクライアントスタブへ返却し、
前記ＲＰＣクライアントスタブは、各サーバオブジェクトのアクセスのための関数呼び出しを発行する時、返却された各サーバオブジェクトのインスタンスＩＤを前記関数呼び出しに含ませる、リモートオブジェクトアクセスシステムにおける、前記サーバコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、リモートプロシージャコール（遠隔手続き呼び出し；以下、「ＲＰＣ」と略称する）を用いて、ネットワークを隔てた別のコンピュータ上のオブジェクトを操作するためのリモートオブジェクトアクセスシステムに関する。
【０００２】なお、ＲＰＣはその実装において、ネットワークからの個々の処理要求を、要求ごとにプロセスを起こし独立したメモリ空間で処理するものと、スレッドやキューイングロジックを用いて共通のメモリ空間で処理するものが存在するが、本方式は後者に適用できる。
【０００３】
【従来の技術】手続き型プログラミングにおける呼び出しをネットワークを隔てたコンピュータ間で実施するために、ＲＰＣが使われている。ＲＰＣはネットワークを隔てた別のコンピュータ上の手続を、あたかもローカルコンピュータ上の手続のように呼出すことができる技術である。ＲＰＣを用いてあるコンピュータが別のコンピュータの手続を呼出すと、その別のコンピュータでその手続が実行され、そして、手続の実行が完了すると、その結果が後者から前者へと返される。
【０００４】
【発明が解決しようとする課題】ところで、オブジェクト指向プログラミングでは、手続型プログラミングにおけるような手続き呼び出しではなく、オブジェクトの生成、処理要求、消滅のようなオブジェクトに対する操作が行われる。このようなオブジェクト指向プログラミングでのオブジェクト操作に関しても、ネットワークを隔てたコンピュータ間で、別のコンピュータ上のオブジェクトをあたかもローカルコンピュータ上のオブジェクトのように扱えることが望ましい。
【０００５】しかしながら、上述した従来のＲＰＣをそのまま利用して、ローカルコンピュータ内でのオブジェクト操作と同様な操作をネットワークを隔てた別のコンピュータ上のオブジェクトに対して行おうと試みた場合、以下のような問題が生じる。
【０００６】図１は、ローカルコンピュータ１内でオブジェクト２を操作する場合の通常のプログラムモデルを

示す。このモデルにおいて、メインプログラム3は次のような操作を行なう。

【0007】1．オブジェクト生成要求を発行し、その結果として、オブジェクトライブラリ4に生成されたオブジェクト2のポインタを取得する。

【0008】2．以後、取得したポインタを使用して、オブジェクト2に対し所望の処理の実行要求を発行する。

【0009】3．オブジェクト2が不要になったら、取得したポインタを使用してオブジェクト2の消滅要求を発行する。

【0010】このようにローカルコンピュータ内の通常のプログラムモデルでは、生成したオブジェクトのポインタを使用して、オブジェクトに対する操作がなされる。

【0011】このようなオブジェクト操作を、従来のRPCを利用して、他のコンピュータ上のオブジェクトに対して行おうとする場合、操作要求はRPCの提供する単純な手続き型言語インタフェースに変換せざるを得ない。そのため、例えば、オブジェクトの生成、処理要求、消滅をそれぞれ手続型言語による手続（関数）として定義し、上述したポインタを関数の引数として設定するような実装を行わざるを得ない。しかし、ネットワークを隔てた別のコンピュータ上にオブジェクトを生成した場合、そのオブジェクトのポインタを取得しても、メモリ空間が異なるために、後の別の関数呼び出しにおいて同じオブジェクトを指定するためにそのポインタを利用できる保証は全くない。つまり、1つの関数呼び出し中ならば、オブジェクトを生成してそのオブジェクトに処理を要求し且つ消滅させることは可能であっても、1つの関数呼び出しでオブジェクトを生成し、別の関数呼び出しでそのオブジェクトに処理を要求し、さらに別の関数呼び出しでそのオブジェクトを消滅させるという操作は、従来のRPCを利用したポインタをやりとりする方式では不可能である。これでは、オブジェクトを必要に応じ動的に生成してアクセスするというオブジェクト指向言語の特長を生かすことができない。

【0012】従って、本発明の目的は、RPCを利用してネットワークを隔てたコンピュータ上にオブジェクトを動的に生成し且つそれにアクセスすることができるリモートオブジェクトアクセスシステムを提供することにある。

【0013】

【課題を解決するための手段】本発明の第1の側面にかかるシステムは、RPCを通じてクライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスする（つまり、処理実行を要求する、及び消滅させる）ためのシステムであって、オブジェクト生成に際しては、クライアントコンピュータが手続型言語インタフェースの生成要求を発し、それに応答してサーバコンピュータがオブジェクトを生成すると共にそのオブジェクトにユニークなIDを割り付け、そのIDをクライアントコンピュータにレスポンスとして返す。以後、サーバコンピュータはオブジェクトのIDとポインタとの対応関係を管理する。そして、クライアントコンピュータは、手続型言語インタフェースのオブジェクトアクセス要求をオブジェクトのIDを指定して発行し、これに応答してサーバコンピュータはそのIDに対応するポインタにより指定されたオブジェクトにアクセスする。

【0014】本発明の第2の側面にかかるシステムは、上記構成に加え、クライアントコンピュータが、サーバコンピュータ内のオブジェクト（サーバオブジェクトという）に対応した仮想的なオブジェクト（クライアントオブジェクトという）を有する。クライアントオブジェクトは、対応するサーバオブジェクトと同一名称のメソッドをサポートし、その生成時及び消滅時には手続型言語インタフェースの生成要求及び消滅要求を発行し、また、サポートするメソッドの要求を受けると、これを手続型言語インタフェースの対応する要求に変換する。サーバコンピュータは、クライアントオブジェクトからの要求に応答して、対応するサーバオブジェクトを生成したりアクセスしたりする。本発明の第3の側面にかかるシステムは、第2の側面のシステムの構成に加え、生成するオブジェクトの種類（クラス）を指定できる機構を更に備える。

【0015】なお、本発明のシステムを構成する各コンピュータのプログラムは、ディスク型ストレージ、半導体メモリ、通信回線などの種々の媒体を通じてコンピュータにインストール又はロードすることができる。

【0016】

【発明の実施の形態】以下、本発明の実施の形態を添付図面に従って説明する。尚、以下の説明では、オブジェクトの操作（生成、処理実行、消滅）要求を発するコンピュータを「クライアントコンピュータ」、その要求を受けてオブジェクトの操作を実行するコンピュータを「サーバコンピュータ」と呼ぶ。

【0017】本発明の第1の実施形態に係るリモートオブジェクトアクセスシステムについて図2～図4を参照して説明する。

【0018】この実施形態の概略は次の通りである。すなわち、サーバコンピュータは、クライアントコンピュータからの生成要求に応答したオブジェクトの生成時に、そのオブジェクトに対してユニークなIDを割り付け、そのオブジェクトのポインタとIDとの対応表を保持すると共に、そのIDをクライアントコンピュータにレスポンスとして返す。クライアントコンピュータは、サーバコンピュータからのIDを保持し、以降のサーバコンピュータ上のオブジェクトに対する操作は、オブジェクトのIDを指定して関数を呼び出すことによって実

施する。

【0019】図2は、この実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータ1内の通常のプログラムモデルを示す。

【0020】図2に示すプログラムモデルは、図1に示した従来のプログラムモデルと比較して明らかなように、メインプログラム3とオブジェクトライブラリ4との間に新たに変換ロジックライブラリ5を有する。この変換ロジックライブラリ5は、オブジェクトの生成、処理実行要求及び消滅というオブジェクト操作ための手続（関数）の集まりであり、メインプログラム3からオブジェクト生成関数の呼び出しを受けると、オブジェクトライブラリ4内にオブジェクト2を生成し、その時にオブジェクト2に対して識別のためのユニークなIDを割り付け、そして、オブジェクトライブラリ4から取得したオブジェクト2のポインタと、オブジェクト2に割り付けたIDとを対応テーブル6に格納して保持及び管理する。

【0021】このプログラムモデルでのオブジェクトの生成とアクセスの動作を以下に説明する。

【0022】1．オブジェクトの生成

メインプログラム3は、オブジェクト生成関数呼び出しを変換ロジックライブラリに発行する。変換ロジックライブラリ5は、オブジェクトライブラリ4内にオブジェクト2を生成し、かつユニークなIDを決定し、そして、取得したポインタとIDとを対応テーブル6に格納し、IDをメインプログラム3にIDを返却する。メインプログラム3は、変換ロジックライブラリ5より生成したオブジェクト2のIDを取得する。

【0023】2．オブジェクトの処理実行

メインプログラム3は、取得したIDを引数にセットしてオブジェクト2に対する処理実行要求関数呼び出しを変換ロジックライブラリ5に発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、そのポインタを使用してオブジェクト2に対して処理実行要求を発行する。

【0024】3．オブジェクトの消滅

メインプログラム3は、オブジェクト2が不要になったら、取得したIDを引数にセットしてオブジェクト消滅関数呼び出しを変換ロジックライブラリ5に発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、そのポインタを使用してオブジェクト2の消滅要求を発行する。

【0025】図3は、図2に示した通常のプログラムモデルの変換ロジックライブラリ5を基に、本実施形態システムのクライアントコンピュータとサーバコンピュータとにそれぞれ搭載されるプログラムユニットを生成する様子を示す図である。

【0026】図2に示す変換ロジックライブラリ5を公知のRPCコンパイラ7に入力すると、変換ロジックのRPCクライアントスタブ（以下、単にクライアントスタブと呼ぶ）8と、変換ロジックのRPCサーバライブラリ（以下、単にサーバライブラリと呼ぶ）9とが自動的に生成される。ここで、クライアントスタブ8とサーバライブラリ9とは、RPC機構によって結合されることによって、全体として図2に示す変換ロジックライブラリ5と同じ機能を果たすものである。クライアントスタブ8は、変換ロジックライブラリ5と同じインタフェースをメインプログラム3に提供するものであり、このインタフェースをRPCの手続型インタフェースに変換する機能をもつ。サーバライブラリ9は、変換ロジックライブラリ5内の対応テーブル6に相当する対応テーブル10を保持及び管理しており、変換ロジックライブラリ5本来の機能、つまり、クライアントスタブ8からRPC機構を通じて到来するオブジェクト操作要求に応じて、実際にオブジェクトを生成し、オブジェクトに処理実行要求を発し、かつオブジェクトを消滅させる機能をもつ。

【0027】図4は、上記のクライアントスタブ8とサーバライブラリ9を搭載したコンピュータにより構成される本実施形態システムのプログラムモデルを示す。

【0028】図4に示すように、クライアントコンピュータ11とサーバコンピュータ12とが、ネットワークを隔てて存在し、両者はRPC機構13によって通信可能である。クライアントコンピュータ11は、メイプログラム3と、図3に示したクラインアントスタブ8とを備える。サーバコンピュータ12は、オブジェクトライブラリ4と、図3に示したサーバライブラリ9を備える。サーバライブラリ9は、前述したようにオブジェクトライブラリ4内の各オブジェクト2のポインタとIDとの対応テーブル10を有する。前述したように、RPC機構13を通じて結合されたクライアントスタブ8とサーバライブラリ9とが、ちょうど図2に示した通常のプログラムモデルの変換ロジックライブラリ5と同様に機能する。RPC機構13を通じてやりとりされるのは、オブジェクトのポインタではなくIDであるから、クライアントコンピュータ11とサーバコンピュータ12とでメモリ空間が異なることは問題にならない。従って、クライアントコンピュータ11はネットワークを隔てたサーバコンピュータ12上のオブジェクト2を操作することができる。

【0029】次に、図4のシステムにおけるオブジェクト操作について説明する。

【0030】1．オブジェクトの生成

クライアントコンピュータ11のメインプログラム3が、クライアントスタブ8に対してオブジェクト生成関数呼び出し発行すると、クライアントスタブ8は、この関数呼び出しをRPC機構13を通じてサーバライブラリ9に送る。サーバライブラリ9は、この関数呼び出しに応答して、オブジェクトライブラリ4内にオブジェク

ト2を生成し、オブジェクトライブラリ4からオブジェクト2のポインタを取得し、オブジェクト2に対しユニークなＩＤを割り当て、そのポインタとＩＤとをテーブル１０に格納し、更に、そのＩＤをＲＰＣ機構１３を通じてクライアントスタブ8へ返却する。このＩＤはクライアントスタブ8からメインプログラム3に渡され、メインプログラム3はそのＩＤを保持する。

【００３１】２．オブジェクトの処理実行

メインプログラム3は、取得したＩＤを引数にセットしてオブジェクト2に対する処理実行要求関数の呼び出しをクライアントスタブ8に発行する。クライアントスタブ8は、その関数呼び出しをサーバライブラリ9に送る。サーバライブラリ9は、その呼び出しに応答して対応テーブル１０からそのＩＤに対応するポインタを取得し、そのポインタによって特定されるオブジェクト2に対して処理の実行要求を発行する。

【００３２】３．オブジェクトの消滅

メインプログラム3は、オブジェクト2が不要になったら、取得したＩＤを引数にセットしてオブジェクト消滅要関数呼び出しをクライアントスタブ8に発行する。クライアントスタブ8は、その関数呼び出しをサーバライブラリ9に送る。サーバライブラリ9は、対応テーブル１０からそのＩＤに対応するポインタを取得し、そのポインタによって特定されるオブジェクト2に対して消滅要求を発行する。

【００３３】以下に、本実施例システムにおける変換ロジックライブラリ5の実装例を示す。オブジェクト指向言語の1つであるＣ＋＋の例えば「ＯｂｊｅｃｔＡ」というクラスについての実装例を説明する。ここで、ＯｂｊｅｃｔＡは、「Ａｃｔｉｏｎ１」「Ａｃｔｉｏｎ２」「Ａｃｔｉｏｎ３」という3つのメソッドを持つものとする。なお、簡単のためＯｂｊｅｃｔＡの生成、消滅及びメソッドにはパラメータがないものとする。

【００３４】変換ロジックライブラリ5を実装する場合、手続型言語であるＣ言語を用いて、オブジェクト操作関数として以下の5つの関数を用意する。

【００３５】１．オブジェクト生成関数「ＯｂｊｅｃｔＡＣｒｅａｔｅ（ＩＤ，ｒｅｔ）」

これはクラスＯｂｊｅｃｔＡのインスタンス（オブジェクト）を生成するための関数である。この関数は、クラスＯｂｊｅｃｔＡのインスタンスを生成すると、そのインズタンスのポインタとそれに割り当てたＩＤとを対応テーブル１０に格納し、そのＩＤと生成結果ｒｅｔを呼び出し元に返却する。

【００３６】２．オブジェクト消滅関数「ＯｂｊｅｃｔＡＤｅｌｅｔｅ（ＩＤ，ｒｅｔ）」

これは、クラスＯｂｊｅｃｔＡのインスタンス（オブジェクト）を消滅させるための関数である。この関数は、渡されたＩＤを基に対応テーブル１０から対応するインスタンスのポインタを取得し、そのポインタにより特定されるインスタンスを削除し、対応テーブル１０からエントリを削除し、その結果ｒｅｔを呼び出し元へ返却する。

【００３７】３．Ａｃｔｉｏｎ１要求関数「ＯｂｊｅｃｔＡＡＣｔｉｏｎ１（ＩＤ，ｒｅｔ）」　これは、メソッドＡｃｔｉｏｎ１をインスタンスに対して要求するための関数である。この関数は、渡されたＩＤを基に対応テーブル１０から対応するインスタンスのポインタを取得し、そのポインタにより特定されるインスタンスに対してＡｃｔｉｏｎ１を発行し、そのＡｃｔｉｏｎ１の結果を結果ｒｅｔとして呼び出し元へ返却する。

【００３８】４．Ａｃｔｉｏｎ２要求関数「ＯｂｊｅｃｔＡＡＣｔｉｏｎ２（ＩＤ，ｒｅｔ）」５．Ａｃｔｉｏｎ３要求関数「ＯｂｊｅｃｔＡＡＣｔｉｏｎ３（ＩＤ，ｒｅｔ）」　発行するメソッドがそれぞれＡｃｔｉｏｎ２、Ａｃｔｉｏｎ３である以外は、ＯｂｊｅｃｔＡＡＣｔｉｏｎ１（ＩＤ，ｒｅｔ）と同じである。

【００３９】以上のＣの関数を図2に示した変換ロジックライブラリ5として実装し、これを図3に示したようにコンパイルしてできたクライアントスタブ8とサーバライブラリ9とを図4に示すようにネットワーク上のコンピュータ１１、１２に搭載することにより、クライアントコンピュータ１１からの要求でサーバコンピュータ１２上にクラスＯｂｊｅｃｔＡのインスタンスを生成し、それにＡｃｔｉｏｎ１、Ａｃｔｉｏｎ２又はＡｃｔｉｏｎ３を要求したり、消滅させたりすることが可能になる。

【００４０】上述した第1の実施形態によると、リモートコンピュータ上にオブジェクトを生成し、処理を要求し、消滅させることができるため、以下のような効果がある。

【００４１】（1）オブジェクトに対する処理要求の結果としてオブジェクト内部に生成した情報をリモートマシン上に保持できる。

【００４２】（2）オブジェクト内部に生成した情報を、独立した処理要求によって取り出すことができる。

【００４３】（3）複数のコンピュータがリモートコンピュータ上のオブジェクトを共有できる。

【００４４】次に本発明の第2の実施の形態に係るリモートオブジェクトアクセスシステムについて図5、図6を参照して説明する。

【００４５】この実施形態の概略は以下の通りである。サーバコンピュータ上のオブジェクト（サーバオブジェクトと呼ぶ）に対応して、それと同じインタフェースをもつオブジェクト（クライアントオブジェクトと呼ぶ）がクライアントコンピュータ上に生成される。ここで、サーバオブジェクトは「真」のオブジェクトであり、一方、クライアントオブジェクトはインタフェースだけがサーバオブジェクトと同じ「仮想的」なオブジェクトで

あるといえる。クライアントコンピュータ内のメインプログラムがクライアントオブジェクトに操作すると、それに対応したサーバオブジェクトの操作がサーバコンピュータ上で実行される。第1の実施形態では、メインプログラムからのオブジェクト操作はC言語のような手続型言語で行う必要があったが、第2の実施形態では、オブジェクト指向言語で行うことができる。

【0046】図5は、この実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータ内の通常のプログラムモデルを示す。

【0047】ローカルコンピュータ21は、図2に示したモデルと同様にメインプログラム3、変換ロジックライブラリ5及びオブジェクトライブラリ4を有する他、メインプログラム3と変換ロジックライブラリ5との間にクライアントオブジェクトライブラリ14を有する。このクライアントオブジェクトライブラリ14は、各サーバオブジェクト2に対応したクライアントオブジェクト15を有する。

【0048】以下、上記モデルでのオブジェクト操作を説明する。

【0049】1．オブジェクトの生成

メインプログラム3が、クライアントオブジェクトライブラリ14内にクライアントオブジェクト15を生成する。クライアントオブジェクト15は、生成処理の中で変換ロジックライブラリ5に対してオブジェクト生成関数の呼び出しを発行する。この呼び出しに応答して変換ロジックのライブラリ5は、サーバオブジェクト2を生成し、オブジェクト2のポインタを取得し、オブジェクト2にユニークなIDを割り付け、ポインタとIDを対応テーブル6に格納し、そして、IDをクライアントオブジェクト15に返却する。クライアントオブジェクト15はそのIDを属性として保持する。

【0050】2．オブジェクトの処理実行

メインプログラム3は、クライアントオブジェクト15に処理の実行を要求する。クライアントオブジェクト15は、属性として保持したIDを引数にセットして、サーバオブジェクト2に対する処理実行要求関数の呼び出しを、変換ロジックライブラリ5へ発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、さおのポインタにより特定されるサーバオブジェクト2に対して処理実行要求を発行する。

【0051】3．オブジェクトの消滅

メインプログラム3は、不要になったクライアントオブジェクト15を消滅させる。クライアントオブジェクト15は、消滅処理の中で、属性として保持したIDを引数にセットしてオブジェクト消滅関数の呼び出し変換ロジックライブラリ5へ発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト2に消滅要求を発行する。

【0052】図6は、図5に示したモデルを基礎にして構築された本実施例システムのプログラムモデルを示す。

【0053】図示のように、クライアントコンピュータ31は、メインプログラム3と、クライアントオブジェクトライブラリ14と、クライアントスタブ8とを有する。サーバコンピュータ32は、サーバライブラリ9と、オブジェクトライブラリ4とを有する。ここで、クライアントスタブ8とサーバライブラリ9は、図3に示したように、変換ロジックライブラリ5を公知のRPCコンパイラでコンパイルすることにより生成されたものである。

【0054】本実施例システムでのオブジェクト操作は以下の通りである。

【0055】1．オブジェクトの生成

メインプログラム3は、オブジェクトライブラリ14内にクライアントオブジェクト15を生成する。クライアントオブジェクト15は、生成処理の中でクライアントスタブ8に対してオブジェクト生成関数呼び出しを発行する。クライアントスタブ18は、その関数呼び出しをRPC機構13を通じてサーバライブラリ19に送る。サーバライブラリ9は、その呼び出しに応答してサーバオブジェクト2を生成し、オブジェクト2のポインタを取得し、オブジェクト2にユニークなIDを割り付け、そのポインタとIDを対応テーブル10に格納し、そしてIDをクライアントスタブ8に返す。クライアントスタブ8はそのIDをクライアントオブジェクト15に渡し、クライアントオブジェクト15はそのIDを属性として保持する。

【0056】2．オブジェクトの処理実行

メインプログラム3は、クライアントオブジェクト15に処理実行を要求する。クライアントオブジェクト15は、属性として保持したIDを引数にセットして処理実行要求関数呼び出しをクライアントスタブ8を通じてサーバライブラリ9に発行する。サーバライブラリ9は、対応テーブル10からIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト2に対して処理実行要求を発行する。

【0057】2．オブジェクトの消滅

メインプログラム3は、不要になったクライアントオブジェクト15を消滅させる。クライアントオブジェクト15は、消滅処理の中で、属性として格納したIDを引数にセットしてオブジェクト消滅関数呼び出しを、RPCクライアントスタブ8を通じてサーバライブラリ9に発行する。サーバライブラリ9は、対応テーブル10からIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト2に消滅要求を発行する。

【0058】以下に、本実施形態システムの実装例を示す。

【００５９】第１の実施形態の実装例と同様に、Ｃ＋＋の「ＯｂｊｅｃｔＡ」をサーバオブジェクトのクラスとし、それが「Ａｃｔｉｏｎ１」「Ａｃｔｉｏｎ２」「Ａｃｔｉｏｎ３」のメソッドを持つものとして、その「ＯｂｊｅｃｔＡ」についての本実施形態の実装例を説明する。

【００６０】まず、第１の実施形態の実装例と同様に、Ｃ言語の５つの関数「ＯｂｊｅｃｔＡＣｒｅａｔｅ（ＩＤ，ｒｅｔ）」「ＯｂｊｅｃｔＡＤｅｌｅｔｅ（ＩＤ，ｒｅｔ）」「ＯｂｊｅｃｔＡＡｃｔｉｏｎ１（ＩＤ，ｒｅｔ）」「ＯｂｊｅｃｔＡＡｃｔｉｏｎ２（ＩＤ，ｒｅｔ）」「ＯｂｊｅｃｔＡＡｃｔｉｏｎ３（ＩＤ，ｒｅｔ）」を、変換ロジックライブラリ５として用意する。

【００６１】次に、サーバオブジェクトのクラスＯｂｊｅｃｔＡに対応したクライアントオブジェクトのクラスＯｂｊｅｃｔＢを用意する。なお、メインプログラム内でＯｂｊｅｃｔＡを使用しない場合、ＯｂｊｅｃｔＢの名称はＯｂｊｅｃｔＡの名称と同一であっても構わない。このＯｂｊｅｃｔＢには、ＯｂｊｅｃｔＡがサポートするメソッドと同一名称のメソッドを用意し、各メソッドが上記のＣ関数を呼び出すように実装する。そのメソッドは以下の通りである。

【００６２】１．コンストラクタ（生成時処理）「ＯｂｊｅｃｔＢ：：ＯｂｊｅｃｔＢ」
これは、生成関数ＯｂｊｅｃｔＡＣｒｅａｔｅ（ＩＤ，ｒｅｔ）を呼び出し、返却されたＩＤを内部に格納する。

【００６３】２．デストラクタ（消滅時処理）「ＯｂｊｅｃｔＢ：：～ＯｂｊｅｃｔＢ」
これは、内部を保持しているＩＤを使用して、消滅関数呼び出しＯｂｊｅｃｔＡＤｅｌｅｔｅ（ＩＤ，ｒｅｔ）を発行する。

【００６４】３．「ＯｂｊｅｃｔＢ：：Ａｃｔｉｏｎ１」
内部保持しているＩＤを使用して、Ａｃｔｉｏｎ１要求関数呼び出しＯｂｊｅｃｔＡＡｃｔｉｏｎ１（ＩＤ，ｒｅｔ）を発行する。

【００６５】４．「ＯｂｊｅｃｔＢ：：Ａｃｔｉｏｎ２」
５．「ＯｂｊｅｃｔＢ：：Ａｃｔｉｏｎ３」
これらは、発行する関数呼び出しがそれぞれＯｂｊｅｃｔＡＡｃｔｉｏｎ２（ＩＤ，ｒｅｔ）、ＯｂｊｅｃｔＡＡｃｔｉｏｎ３（ＩＤ，ｒｅｔ）である以外は、Ａｃｔｉｏｎ１と同じ。

【００６６】以上の第２の実施形態によると、次の利点が得られる。

【００６７】（１）ネットワーク上に配置したいオブジェクトから、機械的にクライアントオブジェクトを生成できるため、簡単にネットワークプログラムが生成できる。

【００６８】（２）ネットワーク上のオブジェクトをローカルコンピュータ内のオブジェクトと同様に、ポインタで制御できる。

【００６９】（３）ローカルコンピュータ内のオブジェクト管理ロジックをリモートコンピュータ上のオブジェクトに対しても適用できる。

【００７０】（４）本実施形態の拡張方式として、クライアントオブジェクトを全くの仮想的なものとせず部分的に実処理を実装することにより、サーバコンピュータとクライアントコンピュータへの負荷分散が実現できる。

【００７１】次に本発明の第３の実施形態に係るリモートオブジェクトアクセスシステムについて、図７、図８を参照して説明する。

【００７２】この実施形態は、上述の第２の実施形態に、操作対象となるオブジェクトの種別（クラス）を任意に指定できる機構を付加したものである。操作対象のオブジェクトのクラスを変更したい場合、第２の実施形態では変換ロジックライブラリを再構築しなければならないが、本実施形態によればその必要がない。

【００７３】図７は本実施形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示す。

【００７４】クライアンコンピュータ４１は、メインプログラム３と、オブジェクトライブラリ５１と、変換ロジックのＲＰＣクライアントスタブ（以下、単にクライアントスタブ）５３とを備える。オブジェクトライブラリ５１には、クライアントオブジェクト５２と、これに割り付けられたＩＤ（以下、インスタンスＩＤという）と、そのオブジェクト種別（クラス）を示す種別ＩＤとが保持される。

【００７５】サーバコンピュータ４２は、変換ロジックのＲＰＣサーバライブラリ（以下、単にサーバライブラリという）５４と、オブジェクトライブラリ５６とを備える。オブジェクトライブラリ５６にはサーバオブジェクト５７が保持される。サーバオブジェクト５７にはクライアントオブジェクト５２が１対１で対応しているが、その具体的な構成は後に説明する。サーバライブラリ５４には、サーバオブジェクト５７のポインタとインスタンスＩＤとの対応テーブル１０、及びサーバオブジェクト５７のオブジェクト種別ＩＤと種類（クラス）との対応テーブル５５とが保持される。

【００７６】クライアントスタブ５３とサーバライブラリ５４は、以下のような変換ロジックライブラリを公知のＲＰＣコンパイラによりコンパイルして生成したものである。この変換ロジックライブラリは次の３つの関数をもつ。

【００７７】（１）オブジェクト生成関数
この関数は、クライアントオブジェクト５２からの呼び出しに含まれているオブジェクト種別ＩＤを基にテーブル５５を参照してそのオブジェクト種別ＩＤに対応した

種類（クラス）を認識し、その種類のサーバオブジェクト57をオブジェクトライブラリ56に生成し、そのオブジェクト57に対しユニークなインスタンスIDを割り当て、そのオブジェクト57のインスタンスIDとポインタとをテーブル10に格納し、更に、そのインスタンスIDをクライアントオブジェクト52に返却する。なお、ここで生成されるサーバオブジェクト57は、後述するように基本サーバオブジェクトから派生しさせて構築したものである。

【0078】（2）オブジェクト消滅関数

この関数は、クライアントオブジェクト52からの呼び出しに含まれているインスタンスIDを基にテーブル10を参照してそのインスタンスIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト57を消滅させる。

【0079】（3）処理要求関数

この関数は、クライアントオブジェクト52からの呼び出しに含まれているインスタンスIDを基にテーブル10を参照してそのインスタンスIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト27に対し、後述する基本サーバオブジェクトがサポートするメソッドを要求する。このメソッドの要求には、後述する処理種別の指定が含まれている。

【0080】図8は、クライアントオブジェクト52とサーバオブジェクト57の構成を示す。

【0081】図8において、基本サーバオブジェクト61は、予め用意された1つのクラスのサーバオブジェクトであり、一つのメソッド60をサポートしている。オリジナルオブジェクト65は、メインプログラム3が生成しようと意図しているオブジェクトであり、ここでは例として3つのメソッド62、63、64をサポートしていることにする。

【0082】図示のようにクライアントオブジェクト52は、オリジナルオブジェクト25から派生させて構築したものであり、構築時にオリジナルオブジェクト25に対して割り付けられたオブジェクト種別IDを属性として持つ。このクライアントオブジェクト52は、図中でオリジナルオブジェクト65と同じ形態に描かれているように、オリジナルオブジェクト65とインタフェースをメインプログラム3に対し提供するものである。このクライアントオブジェクト52は仮想的なオリジナルオブジェクト65ということができ、次のような振舞をする。

【0083】（1）メインプログラム3からの生成要求に応答して生成処理を行い、生成処理では、上述した変換ロジックライブラリに対して、属性として持っているオブジェクト種別IDを引数にセットして上記オブジェクト生成関数の呼び出しを発行し、そして、変換ロジックライブラリから返却されたインスタンスIDを属性として保持する。

【0084】（2）メインプログラム3からオリジナルオブジェクト65のサポートするメソッド62、63又は64の要求を受けると、変換ロジックライブラリに対して、属性として持っているインスタンスIDと、要求されたメソッド62、63又は64に割り付けられた処理種別とを引数にセットして上記処理要求関数の呼び出しを発行する。なお、メソッド62、63、64の処理種別は、クライアントオブジェクト52とサーバオブジェクト57との間で構築時に割り付けられたものである。

【0085】（3）メインプログラムからの消滅要求に応答して消滅処理を行い、消滅処理では、変換ロジックライブラリに対して、属性として持っているインスタンスIDを引数にセットして上記オブジェクト消滅関数の呼び出しを発行する。

【0086】図8に示すように、サーバオブジェクト57は、オリジナルオブジェクト65と基本サーバオブジェクト61の2つのオブジェクトから派生させて構築したものであり、外部に対するインタフェースとして基本サーバオブジェクト61がサポートするメソッド60を有し、このメソッド60の外部から遮蔽された内部に、オリジナルオブジェクト65がサポートするメソッド62、63、64を有し、これらのメソッド62、63、64は前述した処理種別に対応付けられている。

【0087】このサーバオブジェクト57は、変換ロジックライブラリから基本サーバオブジェクト60のサポートするメソッド60を要求されると、その要求に含まれる処理種別に対応したメソッド62、63又は64を実行する。

【0088】以上のモデルにより、メインプログラム3は、自コンピュータ内に所望の種類（クラス）のオブジェクト（オリジナルオブジェクト）を生成するのと同じ操作で、ネットワーク上に当該クラスの真のオブジェクト（サーバオブジェクト57）を生成でき、かつ、自コンピュータ内の所望のオブジェクトに所望のメソッドを要求するのと同じ操作で、ネットワーク上のサーバオブジェクト57に所望のメソッドを実行させることができる。

【0089】本実施形態によれば次の利点が得られる。

【0090】（1）ネットワーク上に配置したいオブジェクトから、機械的にサーバオブジェクト及びクライアントオブジェクトを生成できるため、簡単にネットワークプログラムが生成できる。

【0091】（2）ネットワーク上のオブジェクトをコンピュータ内のオブジェクトと同様に、ポインタで制御できる。

【0092】（3）オブジェクトの種類を追加しない限り、オブジェクトのメソッドの変更などがあったとしても、変換ロジックライブラリを再構築しなくてもよい。

【0093】以上、本発明の幾つかの好適な実施形態を

説明したが、これに変更、修正、改良などを加えた他の種々の形態でも本発明を実施することができる。例えば、図4、6、7に示した実施形態において、メインプログラム3の中にクライアントスタブ8、53を構築することも可能である。

【図面の簡単な説明】

【図1】従来のローカルコンピュータにおける通常のプログラムモデルを示すブロック図である。

【図2】本発明の第1の実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータでの通常のプログラムモデルを示すブロック図である。

【図3】図2に示す変換ロジックライブラリからRPCコンパイラにより変換ロジックのRPCクライアントスタブとRPCサーバライブラリを生成する工程を示す流れ図である。

【図4】第1の実施形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示すブロック図である。

【図5】本発明の第2の実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータでの通常のプログラムモデルを示すブロック図である。

【図6】第1の実施形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示すブロック図である。

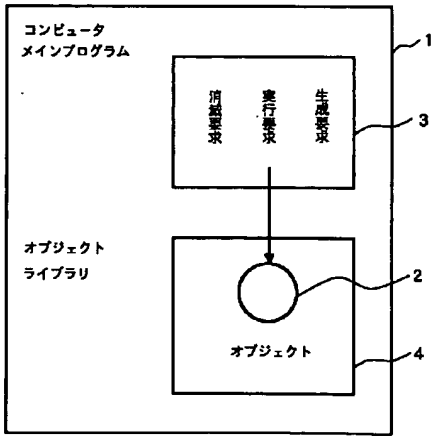
【図7】本発明の第3の実施の形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示すブロック図である。

【図8】第3の実施形態におけるオブジェクトの構成を示す図である。

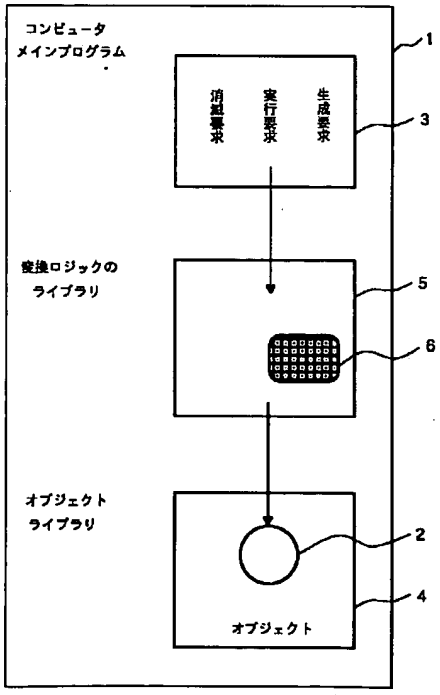
【符号の説明】

2、57　サーバオブジェクト
3　メインプログラム
4、56　オブジェクトライブラリ
5　変換ロジックライブラリ
6、10　オブジェクトのポインタとIDの対応テーブル
8、53　変換ロジックのRPCクライアントスタブ
9、54　変換ロジックのRPCサーバライブラリ
11、31、41　サーバコンピュータ
12、32、42　サーバコンピュータ
13　RPC機構
14、51　クライアントオブジェクトライブラリ
15、52　クライアントオブジェクト
55　オブジェクトの種別IDと種類の対応テーブル

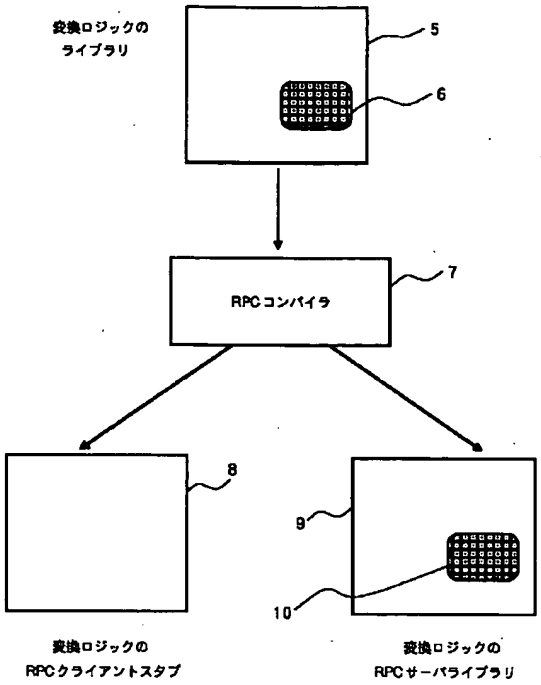
【図1】



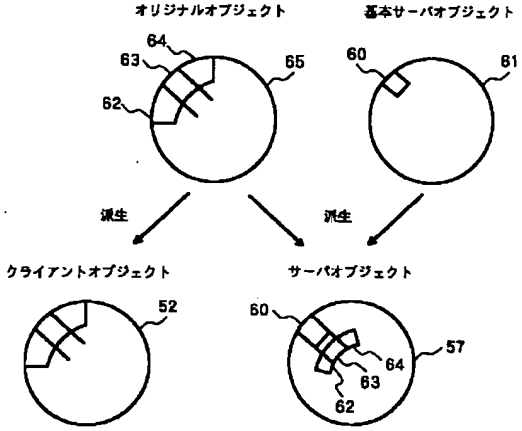
【図2】



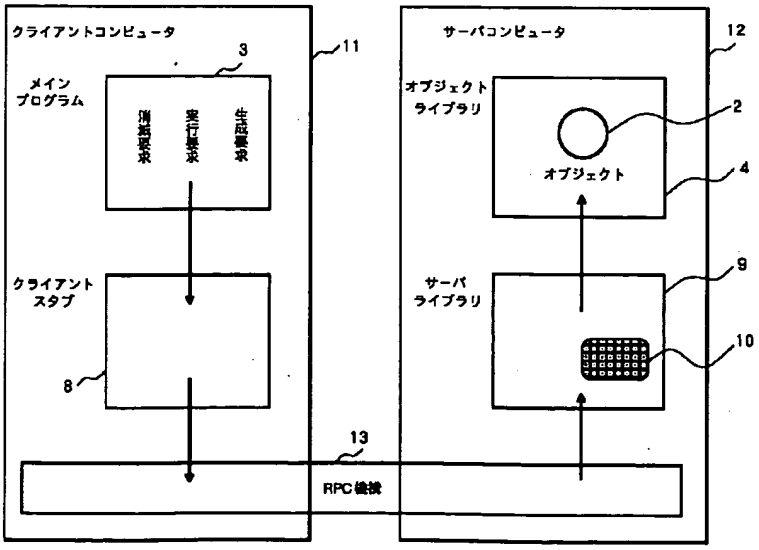
【図3】


変換ロジックの
ライブラリ
5
6
RPCコンパイラ
7
8
9
10
変換ロジックの
RPCクライアントスタブ
変換ロジックの
RPCサーバライブラリ

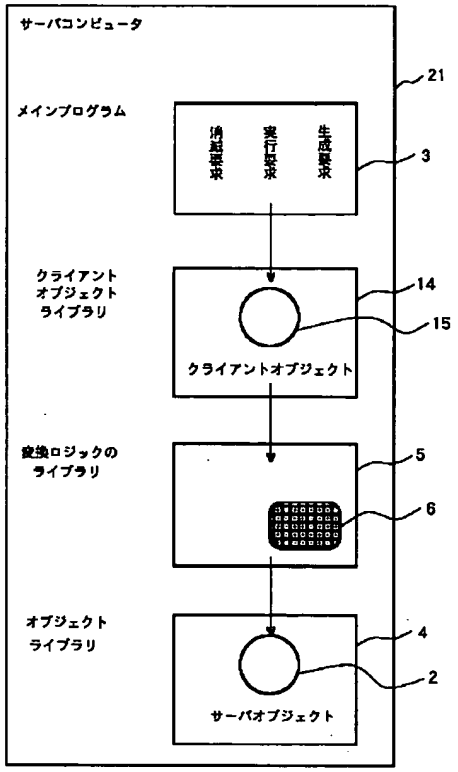
【図8】


オリジナルオブジェクト
基本サーバオブジェクト
64
63
65
62
60
61
派生
派生
クライアントオブジェクト
サーバオブジェクト
52
60
64
57
63
62

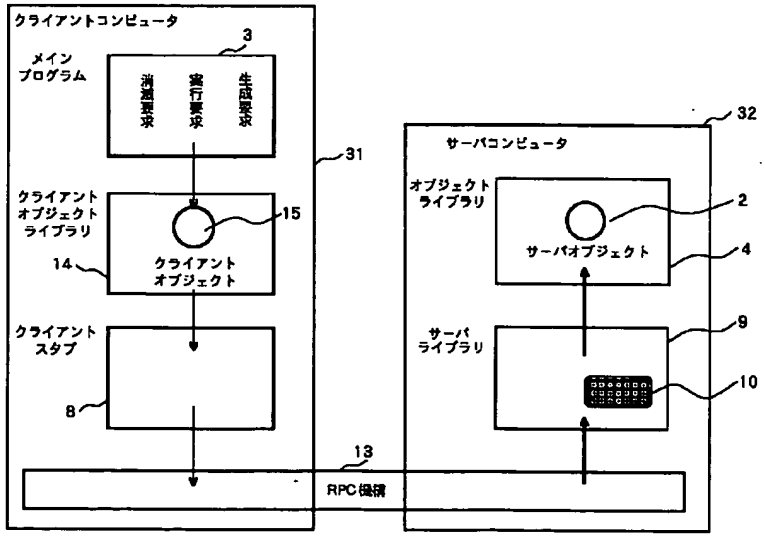
【図4】


クライアントコンピュータ
11
3
メイン
プログラム
消滅要求
実行要求
生成要求
クライアント
スタブ
8
サーバコンピュータ
12
オブジェクト
ライブラリ
2
オブジェクト
4
サーバ
ライブラリ
9
10
13
RPC機構

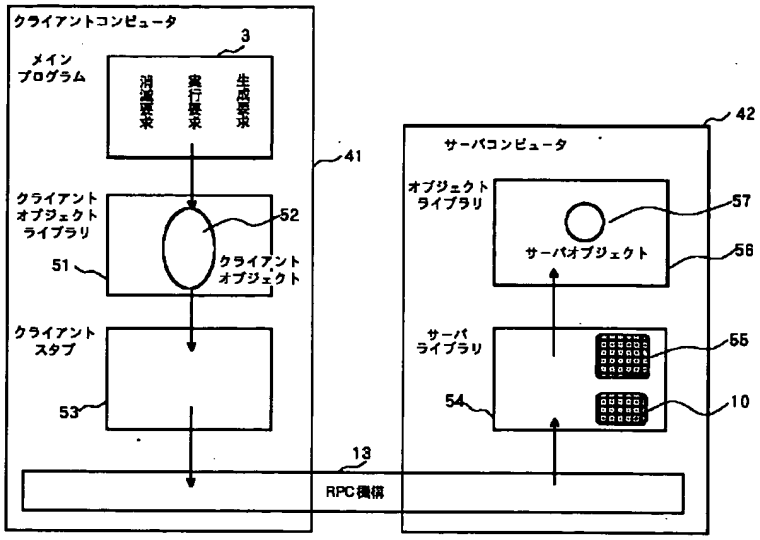
【図5】


サーバコンピュータ
21
メインプログラム
消滅要求
実行要求
生成要求
3
クライアント
オブジェクト
ライブラリ
14
15
クライアントオブジェクト
変換ロジックの
ライブラリ
5
6
オブジェクト
ライブラリ
4
2
サーバオブジェクト

【図6】


クライアントコンピュータ
3
メイン
プログラム
消滅要求
実行要求
生成要求
31
クライアント
オブジェクト
ライブラリ
15
14
クライアント
オブジェクト
クライアント
スタブ
8
13
RPC機構
32
サーバコンピュータ
オブジェクト
ライブラリ
2
サーバオブジェクト
4
サーバ
ライブラリ
9
10

【図7】


クライアントコンピュータ
3
メイン
プログラム
生成要求
実行要求
消滅要求
41
クライアント
オブジェクト
ライブラリ
52
クライアント
オブジェクト
51
クライアント
スタブ
53
13
RPC機構
42
サーバコンピュータ
オブジェクト
ライブラリ
57
サーバオブジェクト
56
サーバ
ライブラリ
55
54
10

PTO 04-1912

Japanese Patent

Document No. H10-275082

REMOTE OBJECT ACCESS SYSTEM AND METHOD

[Remoto Obujekuto Akusesu Shisutemu Oyobi Hoho]

Takashi Yamada

UNITED STATES PATENT AND TRADEMARK OFFICE

Washington, D.C. February 2004

Translated by: Schreiber Translations, Inc.

Country	:	Japan
Document No.	:	H10-275082
Document Type	:	Kokai
Language	:	Japanese
Inventor	:	Takashi Yamada
Applicant	:	NTT Data Communications Co., Ltd.
IPC	:	G 06 F 9/44 9/46 13/00
Application Date	:	February 2, 1998
Publication Date	:	October 13, 1998
Foreign Language Title	:	Remoto Obujekuto Akusesu Shisutemu Oyobi Hoho
English Title	:	REMOTE OBJECT ACCESS SYSTEM AND METHOD

/1[1]

(54) Title of the invention

Remote object access system and method

(57) Summary (amendments included)

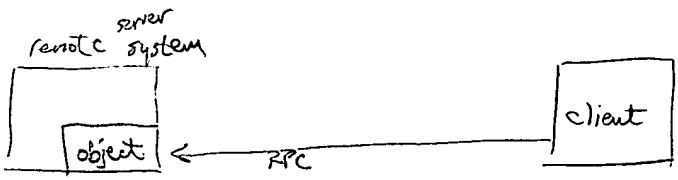
Objective: Attempts are made, by using a remote procedure call (RPC), to generate an object in a dynamic fashion on a computer separated via a network and to access the same.

Solution mechanism: The client computer (11) possesses the main program (3) and the transform logic client stub (8). The server computer (12) possesses the transform logic server library (9). The transform logic server library (9) generates, in response to a generation request issued from the main program (3), the object (2), assigns a unique ID to said object, stores said object ID within the correspondence table (10) together with a pointer, and then returns said object ID to the main program (3). The main program (3) subsequently issues, by designating said object ID, a processing request or deletion request for said object (2). The transform logic server library (9) acquires the pointer corresponding to said object ID from the correspondence table (10) and then processes the object (2) pointed by said pointer.

Patent Claims /2

Claim 1

[1] Numbers in the margin indicate pagination in the foreign text.



A remote object access system with the following characteristics: In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer possesses

An RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer possesses

An RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The RPC client stub and the aforementioned RPC server library exchange, via the aforementioned remote procedure call, unique instance IDs assigned to the respective server objects, whereas

The aforementioned RPC server library possesses a mechanism for transforming the instance IDs of the respective server objects into pointers.

Claim 2

A remote object access method with the following characteristics: In a method which permits a client computer to generate, via the remote procedure call (RPC), an object on a server computer and to access said object,

A process whereby the aforementioned client computer serves the function of an RPC client stub which issues, in response to individual requests for generating and accessing objects generated on the aforementioned client computer, actions for calling individual functions for generating and accessing said objects to the aforementioned server computer via the aforementioned remote procedure call,

A process whereby the aforementioned server computer serves the function of an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates, in response to actions for calling the individual functions received from the aforementioned RPC client stub, server objects on the aforementioned server computer, and which executes the access to said server objects,

A process whereby unique instance IDs assigned to the respective server objects are exchanged, via the aforementioned remote procedure call, between the aforementioned RPC client stub and the aforementioned RPC server library, and

A process whereby the aforementioned RPC server library transforms the instance IDs of the respective server objects into pointers

Are included.

Claim 3

A remote object access system with the following characteristics: In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The aforementioned RPC server library is designed to assign unique instance IDs to the generated server objects, to retain a correspondence table between the instance IDs of the respective server objects and pointers, and to return, to the aforementioned RPC client stub, the instance IDs assigned to the aforementioned generated server objects, whereas

The aforementioned RPC client stub incorporates, into the aforementioned function call, the returned instance IDs of the respective server objects at the time of issuing an action for calling the functions for accessing the respective server objects.

Claim 4

A remote object access system with the following characteristics: In the system specified in Claim 3,

The aforementioned client computer additionally possesses a client object which becomes generated in response to a request for generating the aforementioned object, whereas

The aforementioned client object

Is designed not only

(1): To command, at the time of its generation, the aforementioned RPC client stub to issue an action for calling the function for generating the aforementioned object and to cache the instance ID returned from the aforementioned RPC client stub as a result of the execution of said generation function but also

(2): To command, in response to a request for accessing the aforementioned object, the aforementioned RPC client stub to issue, by using the aforementioned instance ID being cached, a function for gaining the aforementioned access.

Claim 5

A remote object access system with the following characteristics: In the system specified in Claim 4, the aforementioned client object is designed to additionally possess an object type-specific ID which shows the type of said object and

to command, at the time of the generation of said client object, the aforementioned RPC client stub to issue, by using the aforementioned object type-specific ID, a function calling action for generating the aforementioned object, whereas /3

The aforementioned RPC client stub is designed, at the time of issuing a function calling action for generating the aforementioned object, to incorporate the aforementioned object type-specific ID into the aforementioned function call, whereas

The aforementioned RPC server library is designed to additionally possess a correspondence table between object types and object type-specific IDs and to generate, in response to the function calling action for generating the aforementioned object, a server object which is formatted by a method supported by the original object that belongs to the object type corresponding to the aforementioned object type-specific ID.

Claim 6

A remote object access system with the following characteristics: In the system specified in Claim 5,

The aforementioned client object is designed to be branched from the aforementioned original object and to accept a request for the method supported by the aforementioned original object, whereas

The aforementioned server object is designed to be branched not only from the aforementioned original object and a certain foundational server object and to accept a request for the method

supported by the aforementioned foundational server object, whereas

The aforementioned client object is additionally designed, in response to the request for the method supported by the aforementioned original object, to designate a routine type assigned preliminarily to said method, and to command the aforementioned RPC client stub to issue an action for calling the routine request function specific to the method supported by the aforementioned foundational object, whereas

The aforementioned RPC server library is additionally designed, in response to the action for calling the processing request function specific to the method supported by the aforementioned foundational object, to designate the aforementioned routine type and to issue, to the aforementioned server object, a request for the method supported by the aforementioned foundational server object, whereas

The aforementioned server object is additionally designed, in response to the request for the method supported by the aforementioned foundational server object, to execute, among the methods supported by the original object, the method corresponding to the aforementioned routine type.

Claim 7

[A remote object access system with the following characteristics:] In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The RPC client stub and the aforementioned RPC server library exchange, via the aforementioned remote procedure call, unique instance IDs assigned to the respective server objects, whereas

The aforementioned RPC server library is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned client computer within a remote object access system in possession of a mechanism for transforming the instance IDs of the respective server objects into pointers.

Claim 8

[A remote object access system with the following characteristics:] In a remote object access system which permits,

via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The RPC client stub and the aforementioned RPC server library exchange, via the aforementioned remote procedure call, unique instance IDs assigned to the respective server objects, whereas

The aforementioned RPC server library is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned server computer within a remote object access system in possession of a mechanism for transforming the instance IDs of the respective server objects into pointers.

Claim 9

[A remote object access system with the following characteristics:] In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

/4

The aforementioned client computer possesses

An RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The aforementioned RPC server library is designed to assign unique instance IDs to the generated server objects, to retain a correspondence table between the instance IDs of the respective server objects and pointers, and to return, to the aforementioned RPC client stub, the instance IDs assigned to the aforementioned generated server objects, whereas

The aforementioned RPC client stub is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned client computer within a remote object access system designed, at the time of issuing an action for calling the functions for accessing the respective server objects, to incorporate, into the aforementioned function call, the returned instance IDs of the respective server objects.

Claim 10

[A remote object access system with the following characteristics:] In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The aforementioned RPC server library is designed to assign unique instance IDs to the generated server objects, to retain a correspondence table between the instance IDs of the respective server objects and pointers, and to return, to the aforementioned RPC client stub, the instance IDs assigned to the aforementioned generated server objects, whereas

The aforementioned RPC client stub is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned server computer within a remote object access system designed to incorporate, into the aforementioned function calling action, the returned instance IDs of the respective server objects at the time of issuing an action for calling the functions for accessing the respective server objects.

Detailed explanation of the invention

[0001]

(Technical fields to which the invention belongs)

The present invention concerns a remote object access system for operating an object on another computer separated via a network by using the remote procedure call ([transliterated in Japanese]; hereafter abbreviated as the "RPC").

[0002]

Incidentally, the RPC is divided, in terms of loading morphologies, into a format whereby individual processing requests

received from a network are processed within mutually independent memory spaces by initializing processes in request-specific fashions and a format wherein the same are processed within a common memory space by using a thread or cueing logic, whereas the present method is applicable to the latter.

[0003]

(Prior art)

The RPC is being used for exchanging procedural programming calls between computers separated via a network. The RPC represents a technology capable of calling procedures on a computer separated via a network as if they were procedures on a local computer. In a case where a given computer calls for the procedures of another computer by using the RPC, said procedures become executed by said "another" computer, and upon the completion of the execution of said procedures, the results are returned to the latter from the former.

[0004]

(Problems to be solved by the invention)

Coincidentally, in an object-oriented programming context, operations for objects such as the generations of objects, requests for processing the same, and/or deleting the same are executed in place of procedural calls prevailing in the procedural programming context. With regard to such object operations in the object-oriented programming context, too, it is desirable for an

object on a second computer to be handled as if it were an object on a local computer between such computers separated via a network.

[0005]

The following problem, however, is observed in a case where an attempt is made to execute, in relation to an object on a second computer separated via a network, procedures similar to those of object operations within a local computer.

[0006]

Figure 1 shows a normal program model in a case where the object (2) is operated within the local computer (1). According to this model, the main program (3) engages in the following operations.

/5

[0007]

1. An object generation request is issued, as a result of which the pointer for the object (2) generated within the object library (4) is acquired.

[0008]

2. Subsequently, a request for executing a desired routine on the object (2) is issued by using the acquired pointer.

[0009]

3. In a case where the object (2) has become unnecessary, a request for deleting the object (2) is issued by using the acquired pointer.

[0010]

Thus, the normal program model within the local computer is designed to execute the operation of an object by using the generated object pointer.

[0011]

In a case where an attempt is made to execute such an object operation in relation to an object on another computer by using the RPC of the prior art, it is inevitable for the operation request to be transformed into a simple procedural language interface provided by the RPC. For this reason, the generation of the object, requests for processing the same, and the deletion of the same are each defined as procedures (functions) of the procedural language, and the aforementioned pointers must be designated as index numbers of such functions. In a case where an object is generated on another computer separated via a network, however, the memory space is variable, and therefore, even if the pointer for said object is acquired, there is absolutely no guarantee that the same pointer can be used for designating the same object during subsequent actions for calling other functions. In other words, it may be possible, during the progress of the action for calling a singular function, to generate an object, to request a routine for processing said object, and to delete the same, but the format for exchanging a pointer that utilizes the RPC of the prior art is incapable of generating an object based on a single function calling operation, of issuing a request for processing said object by calling another function, and of deleting said object by calling still another function. In such a

case, it is impossible to fully take advantage of the unique advantage of the object-oriented language, namely adventitious generations and accesses of objects in dynamic fashions.

[0012]

The objective of the present invention is therefore to provide a remote object access system which is capable of generating an object on a computer separated via a network by using the RPC in a dynamic fashion and of accessing the same.

[0013]

(Mechanism for solving the problems)

The system pertaining to the first aspect of the present invention is a system which permits, via the RPC, a client computer to generate an object on a server computer and to access said object (i.e., to request the execution of a routine and/or deletion), whereas in the context of generating an object, the client computer issues a request for generating a procedural language interface, in response to which an object is generated by the server computer, and after a unique ID has been assigned to said object, said ID is returned, as a response, to the client computer. The server computer subsequently manages the correspondence relationship between the ID of the object and its pointer. The client computer then issues an object access request for the procedural language interface by designating the ID of the object, in response to which the server computer accesses the object designated by the pointer corresponding to said ID.

[0014]

In addition to being endowed with the aforementioned constitutional attributes, the system pertaining to the second aspect of the present invention is characterized by the fact that the client computer possesses a hypothetical object (referred to as the "client object") corresponding to the object within the server computer (referred to as the "server object"). The client object supports a method which bears a name identical to that of the corresponding server object, whereas during its generation and deletion phases, a request for generating the procedural language interface and a request for deletion are issued, whereas in a case where a request for the supported method has become received, it is transformed into a corresponding request for the procedural language interface. The server computer may, in response to the request issued from the client object, generate and/or access the corresponding server objects. The system pertaining to the third aspect of the present invention is, in addition to being endowed with the constitutional attributes of the system pertaining to the second aspect, further in possession of a mechanism for designating the type (class) of a generated object.

[0015]

Incidentally, the programs of the respective computers that constitute the system of the present invention can be installed or loaded into said computers via various media such as disc-type storage units, semiconductor memories, communications lines, etc.

[0016]

(Application embodiments of the invention)

In the following, application embodiments of the present invention will be explained with reference to attached figures. Incidentally, in subsequent explanations, a computer which issues a request for operations on objects (generation, routine execution, and/or deletion) will be referred to as the "client computer," whereas a computer which executes the operations on objects upon the receptions of such requests as the "server computer."

[0017]

The remote object access system of the first application embodiment of the present invention will be explained with reference to Figures 2 through 4.

[0018]

This application embodiment can be outlined as follows. In other words, the server computer assigns, at the time of the generation of an object in response to a generation request issued from the client computer, a unique ID to said object, retains a correspondence table between the pointers and IDs for such objects, and to return, as a response, the corresponding ID to the client computer. The client computer retains the ID received from the server computer, whereas subsequent operations on the object on the server computer are executed based on function calling actions by designating the ID of the object.

/6

[0019]

Figure 2 shows a normal program model which, within the local computer (1), lays the foundation for building the remote object access system of the present application embodiment.

[0020]

As the comparison with the program of the prior art shown in Figure 1 clearly indicates, the program model shown in Figure 2 additionally possesses the transform logic library (5) between the main program (3) and the object library (4). This transform logic library (5) represents a collection of procedures (functions) for object operations, namely the generations of objects, requests for routine executions, and deletions, whereas in a case where an object generation function call is received from the main program (3), the object (2) is generated within the object library (4), and at the same time, a unique ID is assigned to said object (2) for identification purposes, whereas the pointer for the object (2) acquired from the object library (4) and the ID assigned to the object (2) are stored, retained, and managed in the correspondence table (6).

[0021]

In the following, actions for generating and accessing objects by using this program model will be explained.

[0022]

1. Generation of an object

The main program (3) issues an object generation function call to the transform logic library. The transform logic library (5) proceeds to generate the object (2) within the object library (4), to determine a unique ID, to store the acquired pointer and ID in the correspondence table (6), and then to return said ID to the main program (3) [sic: One extraneous "ID" out of context]. The main program (3) acquires, from the transform logic library (5), the ID for the generated object (2).

[0023]

2. Execution of an object routine

The main program (3) issues, by setting the acquired ID as an index number, a routine execution request function call for the object (2) to the transform logic library (5). The transform logic library (5) acquires the pointer corresponding to the ID from the correspondence table (6) and then issues, by using said pointer, a routine execution request to the object (2).

[0024]

3. Deletion of the object

The main program (3) issues, after the object (2) has become unnecessary, an object deletion function call to the transform logic library (5) by setting the acquired ID as an index number. The transform logic library (5) acquires the pointer corresponding to the ID from the correspondence table (6) and then issues, by using said pointer, a request for deleting the object (2).

[0025]

Figure 3 is a diagram which shows the manners by which program units mounted respectively on the client computer and server computer of the system of the present application embodiment are generated based on the normal program model transform logic library (5) shown in Figure 2.

[0026]

In a case where the transform logic library (5) shown in Figure 2 is inputted into the conventionally-known RPC compiler (7), the RPC client stub for the transform logic (hereafter referred to simply as the "client stub") (8) and the RPC server library for the transform logic (hereafter referred to simply as the "server library") (9) become automatically generated. The client stub (8) and server library (9) hereby serve, in a state where they are mutually being coupled via an RPC mechanism, a collective function similar to that served by the transform logic library (5) shown in Figure 2. The transform logic client stub (8) provides an interface identical to that for the transform logic library (5) to the main program (3) and is endowed with a function of transforming this interface into the procedural interface for the RPC. This server library (9) retains and manages the correspondence table (10), which is equivalent to the correspondence table (6) within the transform logic library (5), and is endowed with the essential functions of the transform logic library (5), namely functions of, in response to object operation requests arriving from the client stub (8) via the RPC mechanism,

actually generating an object, issuing routine execution requests to said objects, and deleting said objects.

[0027]

Figure 4 shows a program model of the system of the present application embodiment constituted by computers loaded respectively with the aforementioned client stub (8) and transform logic server library (9).

[0028]

As Figure 4 indicates, the client computer (11) and the server computer (12) exist apart from one another via a network, whereas both can communicate with one another via the RPC mechanism (13). The client computer (11) possesses the main program (3) and the client stub (8) shown in Figure 3. The server computer (12) possesses the object library (4) and the server library (9) shown in figure 3. The server library (9) possesses, as has been discussed earlier, the correspondence table (10) for the pointers for the respective objects (2) within the object library (4) and their IDs. As has been mentioned earlier, the client stub (8) and server library (9), which are being mutually coupled via the RPC mechanism (13), serves a collective function similar to that of the transform logic library (5) of the normal program model shown in Figure 2. Since it is not the pointer for the object but its ID that is exchanged via the RPC mechanism (13), no problems arise from the mutually different memory spaces of the client computer (11) and the server computer (12). The client computer (11) is therefore capable of operating on the

object (2) on the server computer (12), which is separated via the network.

[0029]

Next, the object operation in the system of Figure 4 will be explained.

[0030]

1. Generation of an object

In a case where the main program (3) of the client computer (11) has issued an object generation function call to the client stub (8), said client stub (8) transmits this function call to the server library (9) via the RPC mechanism (13). The server library (9) generates, in response to this function call, the object (2) within the object library (4), acquires the pointer for the object (2) from the object library (4), assigns a unique ID to the object (2), stores said pointer and ID in the /7 correspondence table (10), and then returns said ID to the client stub (8) via the RPC mechanism (13). This ID is relayed to the main program (3) from the client stub (8), and said ID is retained by the main program (3).

[0031]

2. Execution of an object routine

The main program (3) issues, by setting the acquired ID as an index number, a routine execution request function call for the object (2) to the client stub (8). The client stub (8) transmits

said function call to the server library (9). The server library (9) acquires, in response to said call, the pointer corresponding to said ID from the correspondence table (10) and then issues a routine execution request for the object (2) specified by said pointer.

[0032]

3. Deletion of the object

The main program (3) issues, after the object (2) has become unnecessary, an object deletion function call to the client stub (8) by setting the acquired ID as an index number. The client stub (8) transmits said function call to the transform logic server library (9). The server library (9) acquires a pointer corresponding to said ID from the correspondence table (10) and then issues an deletion request to the object (2) specified by said pointer.

[0033]

In the following, a loading example of the transform logic library (5) of the system of the present application example will be shown. A loading example pertaining to the class of "Object A" of C++, which is an object-oriented language, will be explained. In this context, Object A is presumed to be endowed with three methods, namely "Action 1," "Action 2," and "Action 3." Incidentally, the generation, deletion, and methods of Object A are presumed to possess no parameters for the sake of simplicity.

[0034]

In a case where the transform logic library (5) is loaded, the following five functions are prepared as object operation functions by using the C language, which instantiates a procedural language.

[0035]

1. Object generation function: "Object A Create (ID, ret)"

This is a function for generating an instance (object) of the Object A class. This function stores, upon the generation of an instance of the Object A class, the pointer for said instance and an ID assigned to the same in the correspondence table (10) and returns said ID and generation result ret to the call origination.

[0036]

2. Object deletion function: "Object A Delete (ID, ret)"

This is a function for deleting the instance (object) of the Object A class. This function acquires, based on the relayed ID, the pointer for the corresponding instance from the correspondence table (10), deletes the instance specified by said pointer, deletes the entry from the correspondence table (10), and returns the result ret to the call origination.

[0037]

3. Action 1 request function: "Object A Action 1 (ID, ret)"

This is a function for requesting the instance of Action 1 as a method. This function acquires, based on the relayed ID, the

pointer for the corresponding instance from the correspondence table (10), issues Action 1 to the instance specified by said pointer, and returns the result of said Action 1, as the result ret, to the call origination.

[0038]

4. Action 2 request function: "Object A Action 2 (ID, ret)"

5. Action 3 request function: "Object A Action 3 (ID, ret)"

The procedures are identical to those for Object A Action 1 (ID, ret) except that Action 2 and Action 3 are issued respectively as methods.

[0039]

In a case where the foregoing functions of C are loaded as the transform logic library (5) shown in Figure 2 and where the client stub (8) and the server library (9) are, as Figure 4 shows, loaded respectively into the computers (11) and (12) on the network under the pervasion of the resulting compilation shown in Figure 3, it becomes possible, in response to a request issued from the client computer (11), to generate an instance of the Object A class on the server computer (12), to request the same of Action 1, Action 2, or Action 3, and/or to delete the same.

[0041]

Since it is possible to generate an object on a remote computer, to request a routine for the same, and to delete the same based on the first application embodiment shown above, the following effects can be achieved.

[0041]

(1): As a result of the routine request for the object, it becomes possible to retain, on a remote machine, information generated within the object.

[0042]

(2): The information generated within the object can be retrieved by issuing an independent routine request.

[0043]

(3): The object on the remote computer can be shared by multiple computers.

[0044]

Next, the remote object access system of the second application embodiment of the present invention will be explained with reference to Figures 5 and 6.

[0045]

This application embodiment can be outlined as follows. In correspondence to an object on a server computer (referred to as the "server object"), an object which possesses an interface identical to that of the former (referred to as the "client object") becomes generated on a client computer. The server object hereby represents the "true" object, whereas the client object may, on the other hand, be construed as a "hypothetical" object the interface of which alone is identical to that of the server object. In a case where a main program within the client computer operates on /8

the client object, a corresponding operation on the server object becomes executed on the server computer. As far as the first application embodiment is concerned, the object operation originating from the main program must be executed by using a procedural language such as the C language, whereas an object-oriented language can be used in the second application example.

[0046]

Figure 5 shows a normal main program within a local computer that lays the foundation for building the remote object access system of the present application embodiment.

[0047]

The local computer (21) possesses the main program (3), the transform logic library (5), and the object library (4), as in the case of the model shown in Figure 2, as well as the client object library (14) between the main program (3) and the transform logic library (5). This client object library (14) possesses the client objects (15) corresponding to the respective server objects (2).

[0048]

In the following, the object operations according to the aforementioned model will be explained.

[0049]

1. Generation of an object

The main program (3) generates the client object (15) within the client object library (14). The client object (15) issues, in the midst of the generation routine, an object generation function

call to the transform logic library (5). The transform logic library (5) generates, in response to this call, the server object (2), acquires the pointer for said object (2), assigns a unique ID to said object (2), stores the pointer and ID in the correspondence table (6), and then returns the ID to the client object (15). The client object (15) retains, as an attribute, said ID.

2. Execution of object routine

The main program (3) requests the client object (15) to execute the routine. The client object (15) issues, by setting, as an index number, the ID being retained as an attribute, a routine execution request function call for the object (2) to the transform logic library (5). The transform logic library (5) acquires the pointer corresponding to the ID from the correspondence table (6) and then issues a routine execution request to the object (2) specified by said pointer.

3. Deletion of the object

The main program (3) deletes the client object (15) that has become unnecessary. The client object (15) issues, in the midst of the deletion routine, an object deletion function call to the transform logic library (5) by setting, as an index number, the ID being retained as an attribute. The transform logic library (5) acquires the pointer corresponding to the ID from the

correspondence table (6) and then issues a deletion request to the object (2) specified by said pointer.

[0052]

Figure 6 shows a program model for the system of the present application example built based on the foundation of the model shown in Figure 5.

[0053]

As the figure indicates, the client computer (31) possesses the main program (3), the client object library (14), and the client stub (8). The server computer (32) possesses the server library (9) and the object library (4). The client stub (8) and the server library (9) are, as Figure 3 shows, generated by compiling the transform logic library (5) by using a conventionally-known RPC compiler.

[0054]

The object operation for the system of the present application example is executed according to the following procedures.

[0055]

1. Generation of an object

The main program (3) generates the client object (15) within the client object library (14). The client object (15) generates, in the midst of the generation routine, an object generation function call to the client stub (8). The client stub (18) [sic: Presumably "(8)"] transmits, via the RPC mechanism (13), said function call to the server library (19) [sic: Presumably "(9)"].

The server library (9) generates, in response to said call, the object (2), acquires the pointer for the object (2), assigns a unique ID to the object (2), stores said pointer and ID in the correspondence table (10), and returns the ID to the transform logic client stub (8). The client stub (8) relays said ID to the client object (15), whereas the client object (15) retains said ID as an attribute.

[0056]

2. Execution of object routine

The main program (3) requests the client object (15) to execute a routine. The client object (15) issues, by setting, as an index number, the ID being retained as an attribute, a routine execution request function call to the server library (9) via the client stub (8). The server library (9) acquires the pointer corresponding to the ID from the correspondence table (10) and then issues a routine execution request to the object (2) specified by said pointer.

[0057]

2. [sic: Presumably "3."] Deletion of the object

The main program (3) deletes the client object (15) that has become unnecessary. The client object (15) issues, in the midst of the deletion routine, an object deletion function call to the server library (9) via the RPC client stub (8) by setting, as an index number, the ID being retained as an attribute. The server

library (9) acquires the pointer corresponding to the ID from the correspondence table (10) and then issues a deletion request to the object (2) specified by said pointer.

[0058]

In the following, a loading example of the system of the present application embodiment will be shown.

[0059]

/9

This loading example of the present application embodiment will be explained by designating "Object A" of C++ as a server object class, as in the case of the loading example of the first application embodiment, and by assuming that said "Object A" possesses, as methods, "Action 1," "Action 2," and "Action 3."

[0060]

First, five functions of the C language, namely "Object A Create (ID, ret)," "Object A Delete (ID, ret)," "Object A Action 1 (ID, ret)," "Object A Action 2 (ID, ret)," and "Object A Action 3 (ID, ret)," are prepared as the transform logic library (5), as in the case of the loading example of the first application embodiment.

[0061]

Next, an Object B class is prepared as a client object class corresponding to the Object A class of the server object. Incidentally, the name of Object B may be identical to the name of Object A so long as Object A is not used within the main program.

A method the name of which is identical to that of the method supported by Object A is prepared as this Object B, and the respective methods are loaded for enabling the calls of the aforementioned C functions. These methods are characterized as follows.

[0062]

1. Constructor (generation routine): "Object B : : Object B"

This calls the generation function Object A Create (ID, ret) and stores the returned ID in the system interior.

[0063]

2. Destructor (deletion routine): "Object B : : ~ Object B"

This issues, by using the ID being retained in the system interior, the deletion function call Object A Delete (ID, ret).

[0064]

3. "Object B : Action 1"

The Action 1 request function call Object A Action 1 (ID, ret) is issued by using the ID being retained in the system interior.

4. "Object B : : Action 2"

5. "Object B : : Action 3"

These are identical to Action 1 except that Object A Action 2 (ID, ret) and Object A Action 3 (ID, ret) are respectively issued as function calls.

[0066]

The following advantages are achieved according to the second application embodiment discussed above.

[0067]

(1): Since a client object can be mechanically generated from an object scheduled to be configured on a network, a network program can be generated with ease.

[0068]

(2): An object on the network can be controlled with a pointer according to procedures similar to those for an object within a local computer.

(3): An object management logic within a local computer can also be applied to an object on a remote computer.

(4): As an expanded version of the format of the present application embodiment, an effect of dispersing the load between the server computer and client computer can be realized by partially loading a client object in the course of a real routine instead of construing it as an utterly hypothetical one.

[0071]

Next, the remote object access system of the third application embodiment of the present invention will be explained with reference to Figures 7 and 8.

[0072]

This application embodiment is provided by adding, to the second application embodiment discussed above, a mechanism capable of arbitrarily designating the type (class) of an object selected as an operation target. In a case where one wishes to change the class of an object that prevails as an operation target, it is necessary to reconstruct a transform logic library according to the second application embodiment, whereas such a measure is unnecessary according to the present application embodiment.

[0073]

Figure 7 shows a program model for the remote object access system of the present application embodiment.

[0074]

The client computer (41) possesses the main program (3), the object library (51), and the RPC client stub for the transform logic (hereafter referred to simply as the "client stub") (53). The object library (51) retains the client object (52), an identical assigned to the former (hereafter referred to as the "instance ID"), and a type-specific ID which shows the type (class) of said object.

[0075]

The server computer (42) possesses the RPC server library for the transform logic (hereafter referred to simply as the "server library") (54) and the object library (56). The object library (56) retains the server object (57). The client object (52) corresponds to the server object (57) at a 1 : 1 ratio, and their

concrete constitutions will be explained on a later occasion. The server library (54) retains the correspondence table (10) between the pointer & instance ID of the server object (57) and the correspondence table (55) between the object type-specific ID & type (class) of the server object (57).

[0076]

The client stub (53) and the server library (54) are generated by compiling the following transform logic library by using a conventionally-known RPC compiler. This transform logic library possesses the following three functions.

[0077]

(1): Object generation function

This function acknowledges, with reference to the table (55) based on the object type-specific ID included in the call received from the client object (52), the type (class) corresponding to said object type-specific ID, generates the object (57) of the corresponding type in the object /10 library (56), assigns a unique ID to said object (57), stores the instance ID and pointer for said object (57) in the table (10), and then returns said instance ID to the client object (52). Incidentally, the server object (57) hereby generated is designed to be branched from a foundational server object, as will be discussed on a later occasion.

[0078]

(2): Object deletion function

This function acquires, with reference to the table (10) based on the instance ID included in the call received from the client object (52), the pointer corresponding to said instance ID and then deletes the server object (57) specified by said pointer.

[0079]

(3): Routine request function

This function acquires, with reference to the table (10) based on the instance ID included in the call received from the client object (52), the pointer corresponding to said instance ID and then issues, to the server object (27) [sic: Presumably "(57)"] specified by said pointer, a method supported by the foundational server object, which will be discussed on a later occasion. This method request includes a routine type designation which, too, will be discussed on a later occasion.

[0080]

Figure 8 shows the respective constitutions of the client object (52) and the server object (57).

[0081]

In Figure 8, the foundational server object (61) is a preliminarily prepared server object of a given class, and it supports the singular method (60). The original object (65) is an object which the main program (3) intends to generate, and in this embodiment, three methods (62), (63), and (64) are presumed to be supported.

[0082]

As the figure indicates, the client object (52) is designed to be branched from the original object (25) [sic: Presumably "(65)"], and it possesses, as an attribute, the object type-specific ID assigned to the original object (25) [sic] at the time of system construction. This client object (52), which is illustrated to be morphologically identical to the original object (65) in the figure, provides the original object (65) and an interface to the main program (3). This client object (52), which may be construed as a hypothetical version of the original object (65), engages in the following actions.

[0083]

<1>: It executes a generation routine in response to a generation request received from the main program (3), whereas during said generation routine, it issues the aforementioned object generation function call to the aforementioned transform logic library by setting, as an index number, the object type-specific ID possessed as an attribute, and it then retains, as an attribute, the instance ID returned from the transform logic library.

[0084]

<2>: It issues, upon the reception of a request for the method (62), (63), or (64) supported by the original object (65) from the main program (3), the aforementioned routine request function call to the transform logic library by setting, as index numbers, the instance ID possessed as an attribute and the routine

type assigned to the requested method (62), (63), or (64). Incidentally, the respective routine types assigned to the methods (62), (63), and (64) are assigned between the client object (52) and the server object (57) at the time of system construction.

[0085]

<3>: It executes, in response to a deletion request received from the main program, a deletion routine and issues, during said deletion routine, the aforementioned object deletion function call by setting, as an index number, the instance ID possessed as an attribute.

[0086]

As Figure 8 indicates, the server object (57) is designed to be branched from a pair of objects, namely the original object (65) and the foundational server object (61), whereas it possesses, as an interface with the outside, the method (60) supported by the foundational server object (61), whereas it additionally possesses, in the interior of said method (60) shielded from the outside, the methods (62), (63), and (64) supported by the original object (65), whereas the correspondences of these methods (62), (63), and (64) with the aforementioned routine types are established.

[0087]

This server object (57) executes, upon the reception of a request for the method (60) supported by the foundational server object (60) [sic: Presumably "(61)"] from the transform logic

library, the method (62), (63), or (64) corresponding to the routine type included in said request.

[0088]

It becomes possible, based on the foregoing model, to generate, according to operative procedures identical to those for generating an object (original object) of a certain type (class) within a host computer itself, the true object of the corresponding class [server object (57)] on the network, and to induce the server object (57) on the network to execute, according to procedures identical to those for requesting a desired method in relation to a desired object within the host computer itself, the desired method.

[0089]

The following advantages are achieved based on the present application embodiment.

[0090]

(1): A server object and a client object can be generated mechanically from an object which one wishes to configure on a network, based on which a network program can be generated with ease.

[0091]

(2): An object on the network can be controlled with a pointer in a manner similar to that for an object within a computer.

[0092]

(3): There is no need, so long as no object types are added, to reconstruct a transform logic library even in a case where methods for the object have changed.

[0093]

A handful of desirable application embodiments of the present invention have been explained above, although the present invention can also be implemented based on various other /11 morphologies conceived by altering, modifying, and/or improving them. It is possible, for example, to construct the client stubs (8) and (53) within the main program (3) in the application embodiments shown in Figures 4, 6, and 7.

Brief explanation of the figures

Figure 1: A block diagram which shows a normal program model for a local computer of the prior art.

Figure 2: A block diagram which shows a normal program model for a local computer which lays the foundation for building the remote object access system provided by the first application embodiment of the present invention.

Figure 3: A flow chart which shows a process whereby the RPC transform logic client stub and RPC server library of the transform logic are generated from the transform logic library shown in Figure 2.

Figure 4: A block diagram which shows the program model of the remote object access system of the first application embodiment.

Figure 5: A block diagram which shows a normal program model for a local computer which lays the foundation for building the remote object access system provided by the second application embodiment of the present invention.

Figure 6: A block diagram which shows the program model of the remote object access system provided by the first [sic: Presumably "second"] application embodiment.

Figure 7: A block diagram which shows the program model of the remote object access system provided by the third application embodiment.

Figure 8: A diagram which shows the constitution of the object of the third application embodiment.

(Explanation of notations)

(2) and (57): Server objects;

(3): Main program;

(4) and (56): Object libraries;

(5): Transform logic library;

(6) and (10): Correspondence tables between pointers and IDs of objects;

(8) and (53): RPC client stubs of the transform logic;

(9) and (54): RPC server libraries of the transform logic;

(11), (31), and (41): Server [sic: Presumably "Client"] computers;

(12), (32), and (42): Server computers;

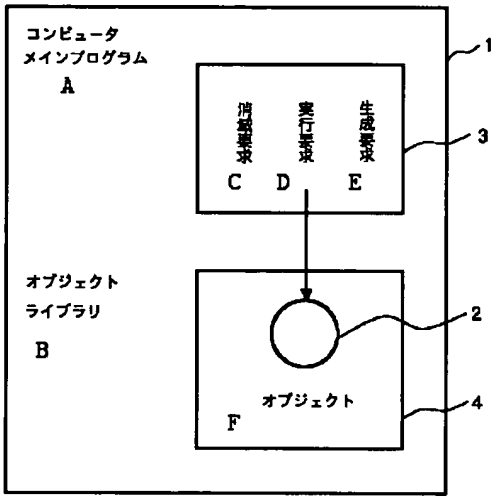
(13): RPC mechanism;

(14) and (51): Client object libraries;

(15) and (52): Client object;

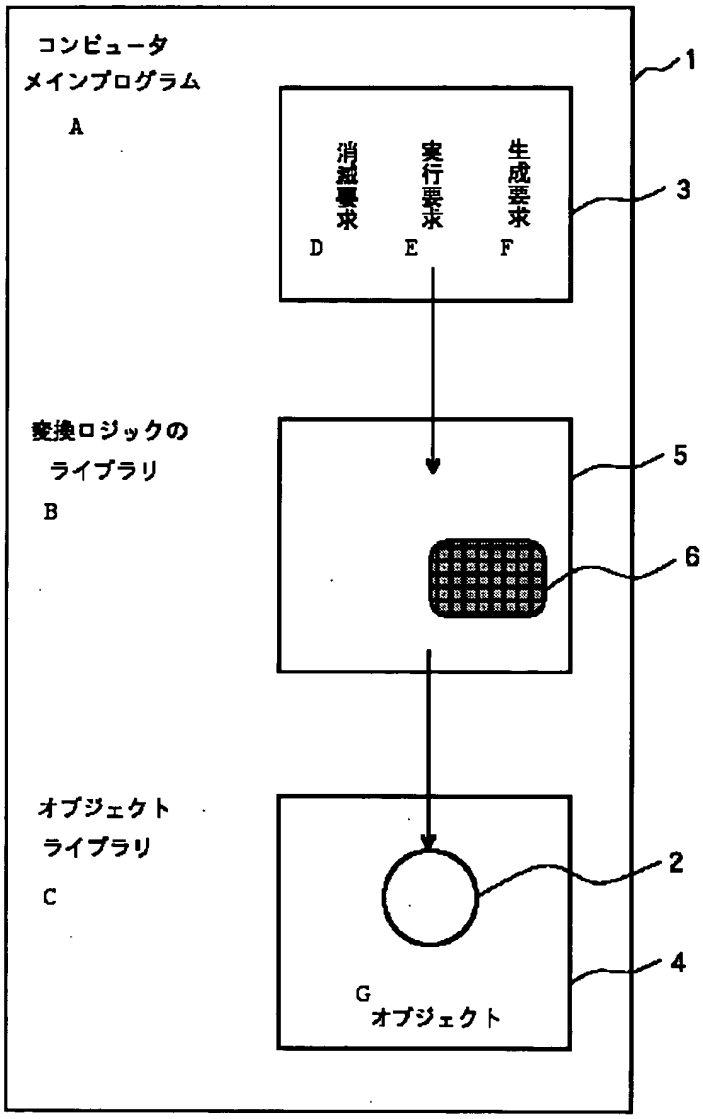
(55): Correspondence table between the type-specific ID and type of an object.

Figure 1



[(A): Computer main program; (B): Object library; (C): Deletion request; (D): Execution request; (E): Generation request; (F): Object]

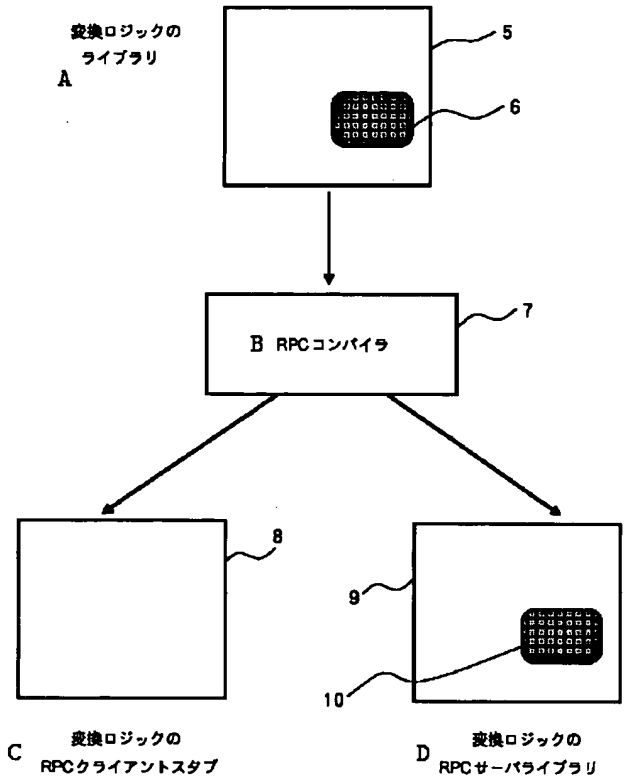
Figure 2



[(A): Computer main program; (B): Library for transform logic; (C): Object library; (D): Deletion request; (E): Execution request; (F): Generation request; (G): Object]

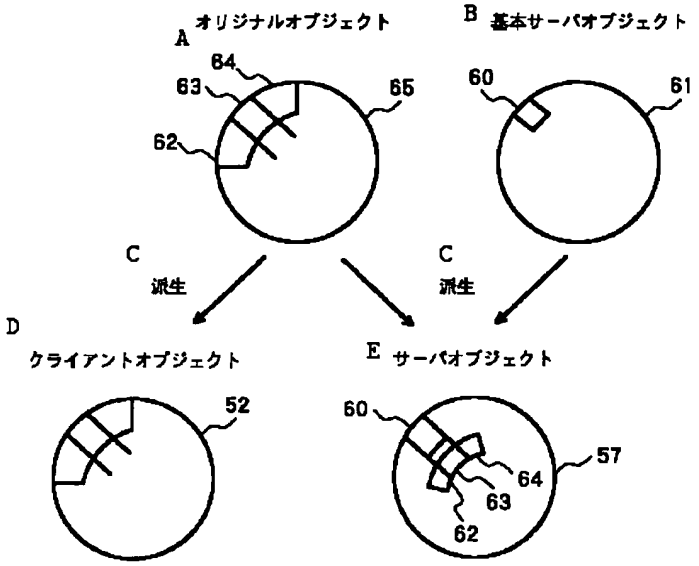
Figure 3

A



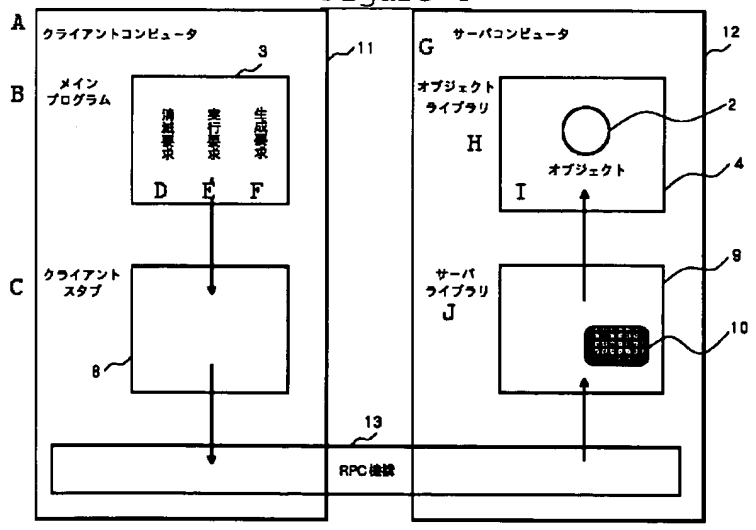
[(A): Library for transform logic; (B): RPC compiler; (C): RPC client stub for transform logic; (D): RPC server library for transform logic]

Figure 8



[(A): Original object; (B): Foundational server object; (C): Branching; (D): Client object; (E): Server object]

Figure 4

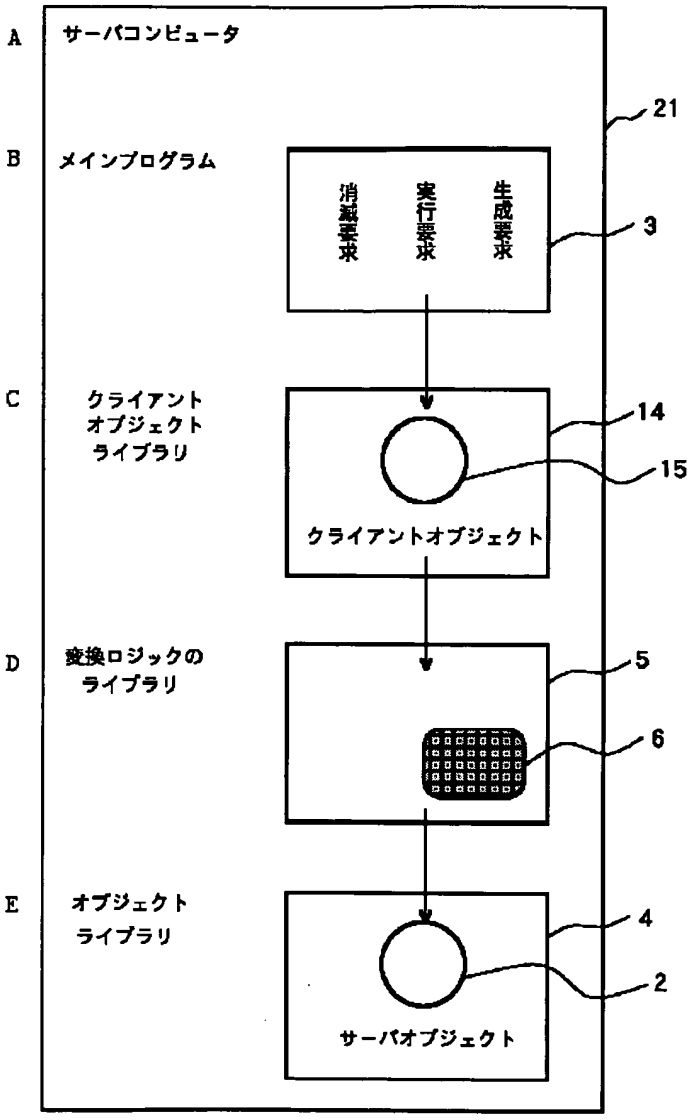


[(A): Client computer; (B): Main program; (C): Client stub; (D): Deletion request; (E): Execution request; (F): Generation request;

(G): Server computer; (H): Object library; (I): Server library; (13): RPC mechanism]

Figure 5

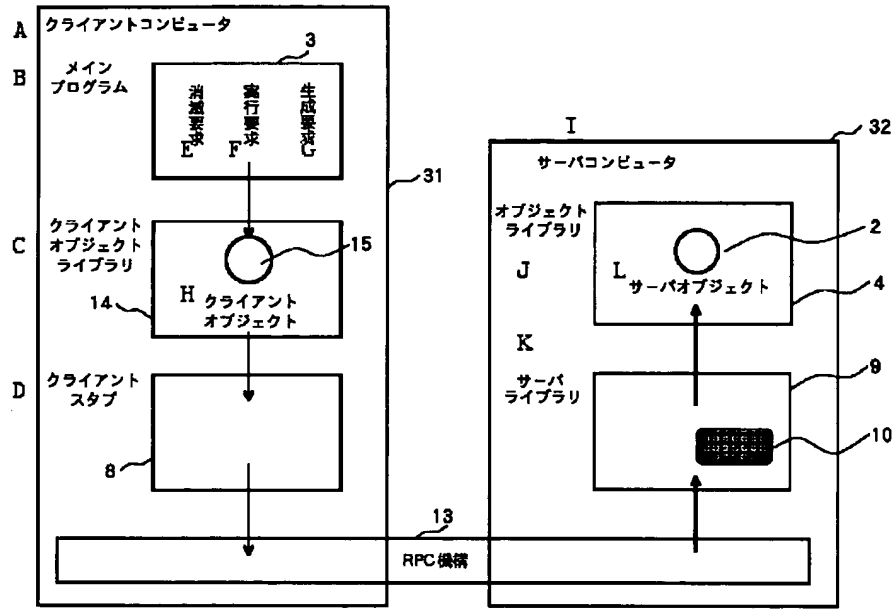
/13



[(A): Server computer; (B): Main program; (C): Client object library; (D): Transform logic library; (E): Object library; (F):

Deletion request; (G): Execution request; (H): Generation request; (I): Client object; (J): Server object]

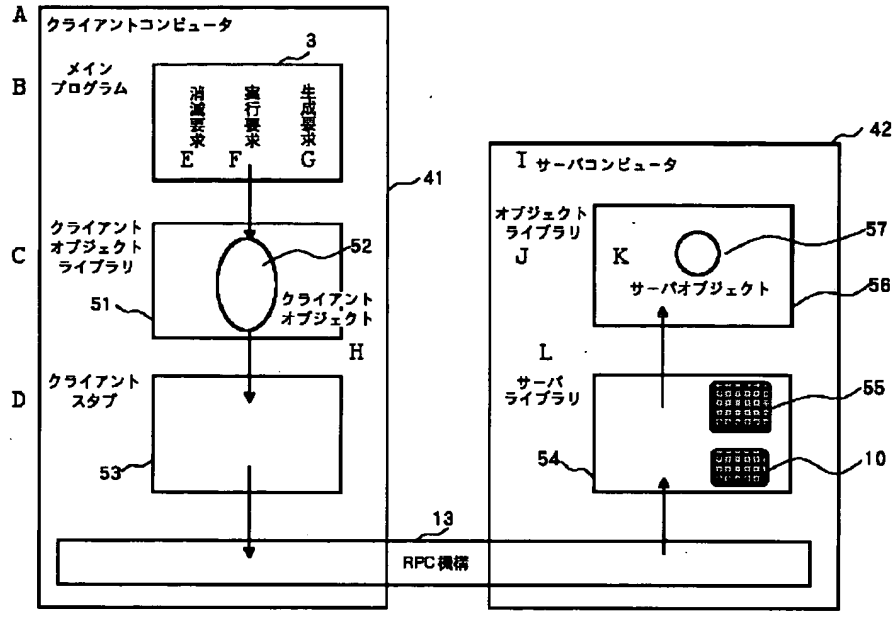
Figure 6



[(A): Client computer; (B): Main program; (C): Client object library; (D): Client stub; (E): Deletion request; (F): Execution request; (G): Generation request; (H): Client object; (I): Server computer; (J): Object library; (K): Server library; (13): RPC mechanism]

Figure 7

/14



[(A): Client computer; (B): Main program; (C): Client object library; (D): Client stub; (E): Deletion request; (F): Execution request; (G): Generation request; (H): Client object; (I): Server computer; (J): Object library; (K): Server library; (13): RPC mechanism]